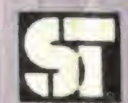
# MSX(2) Basic en machinetaal

W. Duzijn

## De afstand overbrugd

MSX(2) Basic en machinetaal - de afstand overbrugd



# MSX(2) Basic en machinetaal

## De afstand overbrugd

W. Duzijn

uitgeverij STARK-TEXEL b.v.
postbus 302 1794 ZG Oosterend tel. 02223 661

**CIP-gegevens Koninklijke Bibliotheek, Den Haag**

Duzijn, W.

MSX(2) Basic en machinetaal: de afstand overbrugd /
W. Duzijn. - Oosterend : uitgeverij Stark-Texel b.v.
ISBN 90 6398 669 6
SISO 525.5 UDC 800.92 MSX-basic NUGI 851
Trefw.: MSX-basic (programmeertaal).

***eerste druk juni 1987***
**ISBN 90 6398 669 6**

© uitgeverij Stark-Texel b.v.

Niets uit deze uitgave mag worden vermenigvuldigd of openbaar gemaakt worden door middel van druk, fotokopie, mikrofilm of op welke andere wijze dan ook, zonder voorafgaande schriftelijke toestemming van de uitgever.

No part of this book may be reproduced in any form; by print, photoprint, microfilm or any other means, without prior written permission from the publisher.

Ondanks alle aan de samenstelling van de tekst bestede zorg kan noch de auteur noch de uitgever enige aansprakelijkheid aanvaarden voor eventuele schade die zou kunnen voortvloeien uit enige fout die in deze uitgave zou kunnen voorkomen.

# Inhoud

**Inleiding** *7*

**1. Assembler en Basic-interpreter** *9*
1.1 Basic en machinetaal *9*
1.2 Lijst van gebruikte begrippen *13*
1.3 Basic-variabelen en Z80-registers *14*
1.4 Lijst van gebruikte begrippen *20*
1.5 Bytes en bits *21*
1.6 Lijst van gebruikte begrippen *27*
1.7 Logische bewerkingen *28*
1.8 Lijst van gebruikte begrippen *31*
1.9 Nibbles en hexadecimale getallen *31*
1.10 Lijst van gebruikte begrippen *33*

**2. De Z80 microprocessor** *34*
2.1 De Z80 instruktieset *34*
2.2 Lijst van gebruikte begrippen *36*
2.3 De load-instrukties *37*
2.4 Lijst van gebruikte begrippen *41*
2.5 De verwissel-instrukties *42*
2.6 Lijst van gebruikte begrippen *43*
2.7 De rekenkundige of aritmetische instrukties *44*
2.8 Lijst van gebruikte begrippen *47*
2.9 Logische instrukties *49*
2.10 Lijst van gebruikte begrippen *53*
2.11 Roteer- en schuif-instrukties *54*
2.12 Lijst van gebruikte begrippen *58*
2.13 Bit-manipulatie- en bit-test-instrukties *58*
2.14 Lijst van gebruikte begrippen *61*
2.15 Sprong (jump)- en subroutine (CALL/RST)-instrukties *62*
2.16 Lijst van gebruikte begrippen *66*
2.17 Stack-(stapel)instrukties *67*

2.18 Blok-zoek- en verplaats-instrukties *70*
2.19 Input (invoer)- en output (uitvoer)-instrukties *76*
2.20 Enige rest-instrukties *79*

**3. Basic-instrukties worden vertaald** *82*
3.1 De afstand overbrugd *82*
3.2 BEEP *83*
3.3 CHR$ (getal) *83*
3.4 CLS *84*
3.5 COLOR voorgrond,achtergrond,rand *84*
3.6 COPY (kopieer) *85*
3.7 CSRLIN (cursor-line) *86*
3.8 ERL (error-line) *87*
3.9 FOR...NEXT *87*
3.10 GOSUB regelnummer (GOTO SUBroutine) *92*
3.11 GOTO regelnummer *94*
3.12 IF...THEN (als-dan) *95*
3.13 INP poortnummer *98*
3.14 INPUT$(x) *100*
3.15 INTERVAL *102*
3.16 KEY ON/OFF *103*
3.17 LINE INPUT ("tekst") string-variabele *104*
3.18 LOCATE x,y *107*
3.19 LPRINT *107*
3.20 OUT registernummer,te verzenden waarde *108*
3.21 PEEK (adres) *110*
3.22 POKE adres,in te voeren waarde *113*
3.23 PRINT *116*
3.24 PSET(x,y),kleur *119*
3.25 PUT SPRITE spritenummer,(x,y),kleur,patroonnummer *121*
3.26 SCREEN scherminstelling *124*
3.27 SOUND registernummer,registerinhoud *126*
3.28 SPRITE$ (spritenummer) *131*
3.29 STICK (x) *136*
3.30 STOP *137*
3.31 STRIG (x) *137*
3.32 SWAP *138*
3.33 VPEEK (adres) *140*
3.34 VPOKE adres *141*

**4. Basic ROM-routines** *146*
4.1 Inleiding *146*
4.2 ARGDAC: aanroepadres &H2F05 *148*
4.3 ASCDAC: aanroepadres &H3299 *149*
4.4 BASCP: aanroepadres &H2F83 *150*
4.5 BASIC: aanroepadres &H411F *150*
4.6 BASSRL: aanroepadres &H59B4 *151*
4.7 CRLF: aanroepadres &H7328 *152*
4.8 DACARG: aanroepadres &H2F0D *152*
4.9 DACASC: aanroepadres &H3425 *153*
4.10 DACHEX: aanroepadres &H3722 *155*
4.11 DACSGN: aanroepadres &H2EA1 *156*
4.12 DIVDAG: aanroepadres &H4D8B *157*
4.13 ERROR: aanroepadres &H406F *157*
4.14 HLPRT: aanroepadres &H3412 *158*
4.15 LINADR: aanroepadres &H4295 *159*
4.16 MLPDAC: aanroepadres &H3193 *160*
4.17 MULTPL: aanroepadres &H314A *161*
4.18 STRPRT: aanroepadres &H6678 *162*
4.19 VARADR: aanroepadres &H5EA4 *163*
4.20 VARINT: aanroepadres &H5439 *165*

**5. Het vertalen van een Basic programma** *166*
5.1 Inleiding *166*
5.2 Het programmeren van de geluidsgenerator *167*
5.3 SCREEN 2: kleuren veranderen *176*

**6. Testprogramma's** *186*
6.1 Het kontrole-som-programma *186*
6.2 De logische bewerkingen And, Or en Xor *188*
6.3 De bit-manipulatie-instrukties Bit, Set en Res *190*
6.4 Komplement- en tweekomplement (Neg)-bewerkingen *193*
6.5 De schuif-instrukties SRL, SRA en SLA *195*
6.6 De roteer-instrukties RL, RR, RLC en RRC *197*
6.7 De stapel en het vlag-register *200*
6.8 De uitvoer-instruktie 205
6.9 RAM en video-RAM: blokverwerking *207*
6.10 De interruptverwerker *210*
6.11 Zoek en vervang een 'token' in een Basic-regel *213*
6.12 Het plaatsen van een punt op het scherm: PSET *216*
6.13 STICK en STRIG: joystick en vuurknop *218*

6.14 Een eenvoudig sprite-programma *221*
6.15 ASCDAC en VARINT: Basic-ROM-routines *224*
6.16 DACHEX: Basic-ROM-routine *228*
6.17 DIVDAC: Basic-ROM-routine *231*
6.18 LINADR: Basic-ROM-routine *233*
6.19 MLPDAC: Basic-ROM-routine *235*
6.20 VARADR: Basic-ROM-routine *237*

# Inleiding

Er zijn veel computerbezitters die grote interesse hebben voor het interne gebeuren van hun computer, maar die er desondanks nooit toe komen om de in hun ogen grote stap te zetten van het relatief eenvoudige Basic naar het wat duistere en ingewikkelde gebied van het machinetaal-programmeren. Dat is ook wel te begrijpen omdat veel handboeken nogal eenzijdig zijn opgezet, dat wil zeggen: de instruktie-set van de Z80-micro-processor, die het hart is van de MSX-computer, wordt (soms uitvoerig, soms in beknopte vorm) besproken, terwijl niet duidelijk wordt uiteengezet wat het verband is tussen het begrippen-apparaat dat de machinetaal-programmeur gebruikt en de Basic-woordenschat. Vandaar dat dit boek in de eerste plaats een brug-funktie wil vervullen.

Er wordt niet naar volledigheid gestreefd. De opzet is een boek te vervaardigen dat de gebruiker een basis verschaft van waaruit hij zelf verder kan opereren.

Het boek legt uit, licht toe, brengt verbanden aan, geeft veel voorbeelden en kant en klare test-programma's, die niet geschreven zijn om de gebruiker te imponeren, maar om hem in staat te stellen de routines die in het boek worden gesproken binnen een zelfgeschreven toepassingsprogramma te gebruiken.

Op die manier wordt het programmeren een boeiende bezigheid, niet zozeer gericht op resultaat, maar vooral op het verschaffen van een gevoel van voldoening, dat kan ontstaan wanneer men de machine via een machinetaal-routine bepaalde taken kan laten verrichten.

Moeilijke opdrachten worden in het boek niet gegeven De gebruiker krijgt in ruime mate informatie aangereikt en het staat hem vrij die informatie naar eigen inzichten te verwerken.

Het boek is opgebouwd uit drie hoofdonderdelen.
Het eerste deel heeft voornamelijk een leerfunktie: de gebruiker maakt kennis met veelgebruikte begrippen, waarvan een groot aantal in aparte hoofdstukjes worden gedefinieerd. Daarnaast wordt kennisgemaakt met de Z80-instruktie-set. Omdat de Z80 vele honderden instrukties kent is gekozen voor een opdeling in hoofdgroepen. Elke groep instrukties wordt voorafgegaan door een tabel die een globaal overzicht verschaft.

Ook wordt in dit deel verwezen naar een reeks test-programma's die het mogelijk maken inzicht te krijgen in het manipuleren van bits en bytes, met andere woorden: de gebruiker leert de computer kennen als schakelmachine, een simpel apparaat dat weinig meer kan dan het produceren van 'nullen' en 'enen'. Instrukties als BIT, SET en RESet (de zogenaamde bit-manipulatie-instrukties), maar ook de logische bewerkingen AND, OR en XOR worden met behulp van die test-programma's ontdaan van hun duistere gedaante, omdat men nu kan zien wat er gebeurt wanneer dergelijke instrukties worden gebruikt.

Het tweede deel van het boek is in de eerste plaats gericht op de praktijk. Basic-instrukties komen ter sprake, tesamen met de overeenkomstige machinetaal-routines. Tevens worden een aantal ROM-routines die deel uitmaken van de Basic-interpreter besproken, waardoor het mogelijk wordt toepassingsprogramma's te schrijven, die ook relatief moeilijke opdrachten kunnen uitvoeren. Tenslotte worden in dit gedeelte twee Basic-programmaatjes vertaald, dus omgezet in een 'machinetaal-programma'.

Het derde deel van het boek bevat de test-programma's, die deels bestaan uit Basic-programma's die de werking van een aantal Z80-instrukties duidelijk moeten maken, en deels uit gekombineerde 'Basic-machinetaal-programma's', die laten zien hoe bepaalde routines, die in het tweede deel van het boek werden besproken, in een groter programma-geheel kunnen worden opgenomen.

Al met al een boek dat de gebruiker veel geeft: veel informatie, en hopelijk evenveel plezier.

Juni 1987,
W. Duzijn.

# 1. Assembler en Basic-interpreter

## *1.1 Basic en machinetaal*

Een boek dat handelt over de overeenkomsten tussen machinetaal en Basic zal moeten beginnen met een bespreking van de vertaal-programma's, die een adekwaat werken met computer-instrukties mogelijk moeten maken.

Veel gebruikers vergeten wel eens dat hun Basic-programma alleen maar kan werken omdat er in het geheugen een vertaal-programma is opgenomen dat bij de uitvoering van het Basic-bron-programma iedere instruktie omzet in machinetaal. Het programma dat verantwoordelijk is voor die vertaling wordt de Basic-interpreter genoemd. Dit vertaalprogramma wordt bij het opstarten van de computer binnen het bereik van de microprocessor geschakeld, tesamen met een programma dat we het BIOS (dat is Basic Input Output Systeem) noemen.

Deze systeemprogramma's zijn geschreven in machinetaal en ze kunnen met behulp van een uitleesprogramma bekeken worden. Zo'n uitleesprogramma kunnen we zelf op een simpele manier maken met behulp van de Basic-funktie PEEK, bijvoorbeeld:

```
FOR I = 0 TO 200:PRINT PEEK(I):NEXT
```

Alles wat zo'n programma ons echter oplevert is een reeks cijfers, waar we weinig mee kunnen doen. Die cijfers echter vormen de instrukties, die de computer in staat stellen opdrachten uit te voeren.

Wanneer we het geheugengebied van onze computer uitlezen met behulp van het 'PRINT PEEK'-kommando zullen we ontdekken dat dit gebied zich uitstrekt vanaf het getal 0 (we spreken over adres) tot en met het getal (adres) 65535. Dat betekent dat we beschikken over

een geheugengebied dat 65535 geheugenplaatsen omvat (er zijn MSX-computers met een kleiner bereik, maar die laten we in dit boek buiten beschouwing). Zowel de systeemprogramma's (BIOS en Basic-vertaler) als onze zelf geschreven Basic- en/of machinetaal-programma's moeten in dit gebied worden geplaatst.

Wanneer we een Basic-programma intoetsen zijn we ons er nauwelijks van bewust dat ons programma ergens opgeslagen moet worden. De computer doet al het werk voor ons en zorgt ervoor dat een vrij geheugengebied wordt opgevuld met instrukties. Wordt het programma te lang dan volgt een Out of Memory-foutmelding, zodat we maatregelen kunnen nemen om het programma te korrigeren.

Een machinetaalprogramma daarentegen moeten we zelf in het geheugen plaatsen en er is niemand die ons waarschuwt als we een gebied kiezen waarin al programma's staan. Dat plaatsen van machinetaal in het geheugen gebeurt met de assembler. Een assembler heeft net als de Basic-vertaler (interpreter) een bronprogramma nodig. Dat bronprogramma kan op verschillende manieren worden ingevoerd in de computer.

In dit boek wordt gebruik gemaakt van de assembler FLASH, een vertaalprogramma dat gewoon in de Basic-mode werkt. Het bronprogramma kan in Basic-programmaregels worden opgeslagen, waarbij men er op dient te letten dat iedere regel begint met het REM-teken '.

Om het machinetaalprogramma op de juiste plaats in het geheugen te kunnen plaatsen kent de assembler de instruktie: ORG (van Origine). Wanneer we de voorbeeldprogramma's in dit boek bekijken zullen we zien dat ieder machinetaal- (of, juister geformuleerd, assembler-) programma begint met de instruktie ORG 49500. Dat betekent dat vanaf het adres 49500 de instrukties van het machinetaalprogramma worden opgeslagen. Dat gebied mag dus ***niet*** veranderd worden!

De gebruikers van de assembler FLASH hoeven zich daar geen zorgen over te maken. Het programma FLASH reserveert het geheugengebied vanaf het adres 49500 voor eigen gebruik. Op die manier zijn er ongeveer 7 tot 8 duizend geheugenplaatsen vrij voor eigen

programma's. Dat lijkt veel maar in de praktijk zal blijken dat een ruim gebied erg nuttig is om mee te experimenteren.

Wanneer men met een andere assembler werkt zal men er zorg voor moeten dragen dat het gebied vanaf adres 49500 niet wordt aangetast door de Basic-vertaler. Dat kan men bereiken met behulp van het Basic-kommando CLEAR 200,49500. Dit kommando reserveert een gebied van 200 geheugenlokaties of *bytes* voor de opslag van strings en geeft aan de computer de opdracht het adres 49500 als bovengrens (HIMEM) te hanteren.

We kunnen de exakte waarde van die bovengrens (de HIMEM-waarde) achterhalen met behulp van de volgende Basic-opdracht:

```
PRINT PEEK(&HFC4A) + 256*PEEK(&HFC4B)
```

Deze opdracht maakt gebruik van gegevens die door het BIOS-programma en de Basic-vertaler in het geheugen worden opgeslagen vanaf adres &HF380 (decimaal: 62336). Dat gebied wordt het 'werkgeheugen' van de computer genoemd.

Willen we de ondergrens (LOMEM-waarde) bepalen dan moeten we de volgende reeks instrukties intoetsen:

```
PRINT PEEK(&HFC48) + 256*PEEK(&HFC49)
```

De waarde die nu op het scherm wordt geplaatst is het begin-adres van het geheugengebied dat door de Basic-vertaler wordt gebruikt. Wanneer we een programma intoetsen dan zullen de programmaregels vanaf dat adres in het geheugen worden opgeslagen.

We kunnen dit kontroleren met behulp van de volgende PEEK-opdracht:

```
FOR I = 0 TO 100:PRINT PEEK(BEGINADRES + I):NEXT
```

Als beginadres nemen we de LOMEM-waarde die we hierboven hebben uitgelezen. Opnieuw worden we gekonfronteerd met een reeks cijfers, mogelijk afgewisseld met tekst. Het blijkt dat de programmaregels niet letterlijk in het geheugen worden opgeslagen. Dat betekent dat de Basic-vertaler al tijdens het intypen van het pro-

gramma werkzaam is.

Wat gebeurt er dan precies? Wel, eenvoudig gezegd komt het hier op neer: gedurende het intypen van de programma-tekst wordt de informatie tijdelijk opgeslagen in een aantal vrije bytes in het werkgeheugen. Zo'n gebied van vrije geheugenlokaties in het werkgebied van de computer noemen we een 'buffer'.

Drukken we op de Enter-toets (om aan te geven dat de programmaregel voltooid is) dan zet de Basic-vertaler de tekst in die buffer om in een reeks koderingen en deze gekodeerde 'tekst'-regel wordt in het geheugen geplaatst. Wanneer we derhalve het geheugen uitlezen dan zien we deze gekodeerde regel. Een gekodeerde Basic-instruktie wordt TOKEN genoemd. Zo is de kode voor het Basic-kommando PRINT het getal 145. De kode voor het kommando LPRINT is 157*.

Het bronprogramma voor de assembler FLASH wordt door de Basic-vertaler niet in tokens omgezet omdat iedere programmaregel moet beginnen met het REM-teken '. Dit teken vertelt de computer dat de tekst die er op volgt letterlijk in het geheugen moet worden opgeslagen.

De assembler werkt dus met gewone tekstregels, die tijdens het vertalingsproces direkt worden omgezet in machinetaal. Wanneer de assembler het bron-programma heeft vertaald kan het feitelijke machinetaal-programma in zijn geheel worden bewaard door opslag in een aantal Basic-dataregels of door het weg te schrijven naar disk met behulp van het Basic-BSAVE-kommano.

Het voordeel van deze procedure is dat het machinetaal-programma direkt, zonder dat er een vertaal-procedure nodig is, kan worden verwerkt wanneer we het aanroepen met behulp van de Basic-USR-

---

* Let op: voor een volledige lijst van TOKENS wordt verwezen naar het MSX Zakboekje van Wessel Akkermans, een handig naslagwerk dat in kompakte vorm alle informatie bevat die we bij het programmeren in Basic en machinetaal nodig hebben (uitgeverij Stark-Texel b.v.).

funktie.

Het Basic-bronprogramma echter moet altijd, tijdens iedere uitvoering van het programma, stap voor stap vertaald worden. Dat brengt voordelen met zich mee, in die zin dat er een ruime foutenkorrektiemogelijkheid voorhanden is, waarbij slechts een programmaregel veranderd hoeft te worden wanneer er een fout wordt ontdekt. Helaas blijft het nadeel van de trage werking van het programma. Hoe meer stappen een computer moet zetten des te meer tijd heeft hij nodig - dat is een wet waar niet aan te tornen valt. Ook wanneer we programmeren in machinetaal zullen we daar op moeten letten.

## *1.2 Lijst van gebruikte begrippen*

**Basic-vertaler (interpreter)**

Vertaalprogramma dat Basic-instrukties in twee stappen vertaalt:

1. tijdens het intypen worden tekstregels omgezet in Basic-kode-regels;

2. tijdens de uitvoering van het programma worden de gekodeerde instrukties stuk voor stuk omgezet in machinetaal (d.w.z. er worden machinetaal-routines aangeroepen die deel uitmaken van het ingebouwde MSX-Basic-systeem-programma).

Plaats in het geheugen: adressen 16384 - 32767.

(De MSX2-computers hebben een uitgebreid Basic-ROM, waar hier niet verder op in wordt gegaan; wij beperken ons tot die Basic-routines, die zowel op de MSX1 als de MSX2-computers werkzaam zijn.)

**Assembler**

Vertaalprogramma dat een reeks assembler-instrukties (ook wel mnemonics genoemd) omzet in machinetaal. De vertaling hoeft maar één keer te worden uitgevoerd.

**BIOS (Basic Input Output Systeem)**

Systeem-programma, dat is opgebouwd uit een aantal standaardroutines, die het werken met de randapparatuur vergemakkelijken. Deze routines worden gebruikt door de Basic-interpreter, maar kunnen ook in toepassingsprogramma's worden opgenomen.

Plaats in het geheugen: adressen 0 - 16384.

**Byte**

Een lokatie in het geheugen dat een getal kan bevatten. Dit getal heeft veelal een symbolische betekenis - d.w.z. het verwijst naar een bepaald gegeven, zoals een Basic-instruktie, een machinetaal-instruktie, een bepaalde letter of iets dergelijks. De praktijk wijst uit dat dit getal niet groter mag zijn dan 255.

**Adres**

Getal dat verwijst naar een geheugenlokatie of byte.

**Token**

Kode-getal, dat verwijst naar een Basic-instruktie.

## *1.3 Basic-variabelen en Z80-registers*

Wanneer men een Basic-programma schrijft gebruikt men niet alleen de Basic-instrukties. Een Basic-programma kan alleen geschreven worden wanneer men de beschikking heeft over de mogelijkheid gegevens vast te houden, zodat ze voor later gebruik beschikbaar zijn. Hetzelfde geldt voor een machinetaal-programma!

Het feitelijke besturingsorgaan van de MSX-computer, de Z80-microprocessor, kent maar een beperkt aantal opslagmogelijkheden. Dat betekent dat zowel de Basic-programmeur als de machinetaal-programmeur het RAM-geheugen (RAM staat voor vrij geheugen, in tegenstelling tot ROM: geheugen dat alleen uitgelezen kan worden) zullen moeten gebruiken als opslaggebied.

Het Basic-vertaalprogramma reserveert daarvoor tijdens de uitvoering van een programma een geheugengebied dat onmiddellijk volgt

op het Basic-programma. Dat gebied bestaat uit twee delen: het variabelengeheugen en het array geheugen (zie figuur 1.3a).

```
SITUERING VAN HET BASIC-PROGRAMMA IN HET RAM-GEHEUGEN
---------------------------------------------------------
Het start-adres van het RAM-gebied dat voor programma's
beschikbaar is staat in de systeemvariabele BOTTOM
(&HFC48) (normaliter is dat het adres 32768)
---------------------------------------------------------
Adres van het eerste byte van het programmageheugen (het
startadres van het Basic-programma) staat in de systeem-
variabele TXTTAB (&HF676)
(normaliter is dat het adres 32768)
---------------------------------------------------------
Adres van het begin van het variabelen-geheugen (dat is
tevens het einde van het Basic-programma) staat in de
systeemvariabele VARTAB (&HF6C2)
---------------------------------------------------------
Adres van het begin van het array-geheugen (tevens het
eind van het variabelen-gebied) staat in de systeemvari-
abele ARYTAB (&HF6C4)
---------------------------------------------------------
Adres van het einde van het array-geheugen (tevens begin
van het vrije RAM-gebied) staat in de systeemvariabele
STREND (&HF6C6)
---------------------------------------------------------
```

*figuur 1.3a*

Een variabele is dus in feite een geheugengebiedje waarin informatie opgeslagen kan worden. We kennen numerieke variabelen, die getallen kunnen bevatten, waarmee berekeningen kunnen worden uitgevoerd, en we kennen string-variabelen, waarin tekst wordt opgeslagen (of getallen in 'tekst-vorm', waarmee geen berekeningen kunnen worden uitgevoerd).

De numerieke variabelen kunnen op hun beurt weer onderverdeeld worden in integer-variabelen (variabelen die alleen gehele getallen kunnen bevatten), variabelen met enkele precisie (kunnen maximaal 6 cijfers bevatten) en variabelen met dubbele precisie (maximaal 14 cijfers). Alles wat we bij het programmeren in Basic hoeven te doen is het toekennen van de gewenste waarde aan een variabele.

Bijvoorbeeld:

```
A%=124
```
*of*
```
B#=1200.1234.
```

De Basic-vertaler zet de getallen die we intoetsen om in een getallen-kode die de computer kan begrijpen (zie figuur 1.3b) en plaatst die getallen in het variabelen en/of arraygeheugen.

```
KODERING VAN GETALLEN DOOR DE BASIC-INTERPRETER
-----------------------------------------------------------
1  Numerieke konstanten:

   0 tot en met 9      1 byte:  kodenrs 17 tot en met 26
   10 tot 256          2 bytes: prekode 15; getal 1 byte
   256 tot 32767       3 bytes: prekode 28; getal 2 bytes
   Enkele precisie     5 bytes: prekode 29; getal 4 bytes
   Dubbele precisie    9 bytes: prekode 31; getal 8 bytes
   Hexadecimaal getal  3 bytes: prekode 12; getal 2 bytes
   Oktaal getal        3 bytes: prekode 11; getal 2 bytes
-----------------------------------------------------------
2  Regelnummers en regelpointers:

   Instrukties als GOTO, GOSUB e.d. hebben een eigen
   kodering. Tijdens het intypen wordt een regelnummer
   achter de instruktie geplaatst, voorafgegaan door het
   kodenummer 14. Tijdens de uitvoering van het program-
   ma wordt het regelnummer omgezet in een adres-wijzer
   (pointer), voorafgegaan door het kodenummer 13 (er
   wordt nu verwezen naar het startadres van de regel)

   Regelnummer         3 bytes: prekode 14; getal 2 bytes
   Pointer             3 bytes: prekode 13; getal 2 bytes
-----------------------------------------------------------
```
*figuur 1.3b*

Om de kodering van getallen uit te testen kunnen we de volgende regels invoeren:

```
10 PRINT 12 + 1200
20 FOR I=32769 TO 32783
30 PRINT PEEK(I);:NEXT
```

RUN-nen geeft het volgende resultaat:

```
2512
16 128 10 0 145 32 15 12 32 241 32 28 176 4 0
```

De eerste vier getallen bevatten het adres van de volgende regel (lage byte:16 - hoge byte:128 - de hoge byte moet met 256 worden vermenigvuldigd om het juiste getal te verkrijgen!) en het regelnummer (lage byte:10 - hoge byte:0).

| | |
|---|---|
| 145 | het token voor het Basic-kommando PRINT. |
| 32 | de ASCII-kode voor een spatie. |
| 15 | de prekode van het getal 12. |
| 12 | getal. |
| 32 | spatie. |
| 241 | het token voor het + teken. |
| 32 | spatie. |
| 28 | de prekode van het getal 1200 |
| 176 | lage byte getal |
| 4 | hoge byte getal (vermenigvuldigen met 256). |
| 0 | is een eindmarkering (geeft het einde van de regel aan). |

Om de toekenning van een waarde aan een variabele uit te testen toetsen we de volgende regels in (zie figuur 1.3a):

```
10 A%=12:B%=1200
20 VS!=PEEK(&HF6C2) + 256*PEEK(&HF6C3)
30 VE!=PEEK(&HF6C4) + 256*PEEK(&HF6C5)
40 FOR I=VS! TO VE!
50 PRINT PEEK(I);:NEXT
```

Runnen geeft het volgende resultaat (voor de duidelijkheid splitsen we de gegevens hier uit):

**Variabele A%:**

2 (integerkode) 65 (ASCII-waarde A) 0 (variabele mag uit twee

tekens bestaan - 2e teken wordt hier niet gebruikt) 12 (lage byte getal) 0 (hoge byte getal)

**Variabele B%:**

2 (kode) 66 (naam=B) 0 (2e teken) 176 (lage byte) 4 (hoge byte)

**Variabele VS!:**

4 (enkele precisie-kode) 86 (ASCII-waarde V) 83 (2e teken S) 69 50 135 48 (getal in enkele precisie- of BCD-kode)

**Variabele VE!:**

4 (kode) 86 (V) 69 (E) 69 50 137 112 (getal in kode)

We zien dus dat het variabelengebeugen door de Basic-interpreter gevuld wordt met de gegevens, die we hebben ingetoetst.

Wanneer we een machinetaal-programma willen schrijven zullen we ook variabelen nodig hebben waarin we al of niet tijdelijk gegevens kunnen opslaan.

Die variabelen zullen we voor een groot deel zelf moeten ontwerpen, d.w.z. we moeten zelf geheugenplaatsen reserveren en we zullen zelf moeten onthouden welke geheugenplaatsen voor welk doel gebruikt worden. We kunnen daarbij niet slordig of nonchalant te werk gaan, omdat elke fout wordt bestraft met een vastlopen van de machine (er is immers geen vertaal-programma werkzaam dat eerst de ingevoerde informatie kontroleert, zoals dat wel het geval is wanneer we een Basic-programma uitvoeren).

De Z80-microprocessor, waar we rechtstreeks mee te maken krijgen bij het programmeren in machinetaal, kent een aantal 'vaste variabelen', een wat onlogische term, waarmee we de Z80-registers willen aanduiden (zie figuur 1.3c).

```
--------------------------------------------------------
Z80 REGISTERS EN HULPREGISTERS
--------------------------------------------------------
Hoofdregisters                              Hulpregisters

A of akku       F of vlag-register          A'   -   F'
B               C                           B'   -   C'
D               E                           D'   -   E'
H               L                           H'   -   L'
--------------------------------------------------------
INDEX-registers: IX en IY                16-bits registers

Met behulp van de assembler FLASH kunnen ook afzonder-
lijke helften (high-bytes en low-bytes) worden beinvloed
--------------------------------------------------------
STACK-POINTER of SP-register

16-bits register, dat het aktuele adres van de Stack of
Stapel bevat
--------------------------------------------------------
```

*figuur 13c*

Deze registers kunnen we beschouwen als integere variabelen, d.w.z. er kunnen alleen gehele getallen in worden geplaatst. Werken met 'getallen achter de komma' (b.v. 12.123) is niet mogelijk, tenzij men overgaat tot het schrijven van ingewikkelde routines (hetgeen buiten de strekking van dit boek valt).

Het aantal registers is, zoals we hierboven kunnen zien, beperkt. Dat is een groot verschil met Basic, waar we over een onbeperkt aantal variabelen (slechts beperkt door de geheugenruimte) kunnen beschikken. Het goed werken met machinetaal vereist daarom dat we moeten leren hoe we deze registers meerdere malen kunnen gebruiken (via de mogelijkheid van de Z80-processor registerinhouden tijdelijk in het geheugen te plaatsen). We komen hier later nog op terug.

De belangrijkste registers zijn het A-register, dat ook wel de akku of de akkumulator wordt genoemd, en het F- of vlag-register, waarin na afloop van rekenkundige en logische bewerkingen bepaalde informatie wordt geplaatst, die we voor eigen gebruik uit kunnen

lezen.

Vrijwel alle berekeningen worden uitgevoerd met behulp van het A-register. Dat betekent dat we dit register zelden als 'variabele' zullen kunnen gebruiken. Een variabele moet een waarde vast kunnen houden, zodat we, iedere keer wanneer dat nodig is, die waarde op kunnen roepen. De andere registers (behalve het SP-register), die wel als 'variabele' toegepast kunnen worden, worden binnen een programma op verschillende manieren gebruikt:

a. als 8 bits-register,
b. in kombinatie met een ander register als 16 bits-register.

Daarmee belanden we bij het begrip BIT. In deel 5 van dit hoofdstuk zullen we een uitvoerige bespreking wijden aan dit begrip.

## *1.4 Lijst van gebruikte begrippen*

**Z80-micro-processor**

Dit is het hart van de MSX-computer. Zowel Basic-programma's als zelfgeschreven machinetaal-programma's geven hun informatie door aan dit centrale rekenorgaan. De processor kan rechtstreeks geprogrammeerd worden met behulp van een reeks assembler-instrukties (mnemonics) die door de assembler worden omgezet in machinetaal.

**RAM (Random Access Memory)**

Geheugen waarin de gebruiker vrijelijk gegegevens kan opslaan.

**ROM (Read Only Memory)**

Geheugen dat alleen kan worden uitgelezen. Voor alle MSX-computers geldt dat het gebied tussen adres 0 en adres 32768 bezet wordt door ROM-systeemprogramma's.

**Variabele**

Lokatie waarin bepaalde gegevens kunnen worden opgeslagen. Tij-

dens de uitvoering van een Basic-programma wordt een gebied gereserveerd voor variabelen. Bij het ontwerpen van machinetaal-programma's zal men zelf de benodigde 'lokaties' moeten reserveren.

### Akku (akkumulator of A-register)

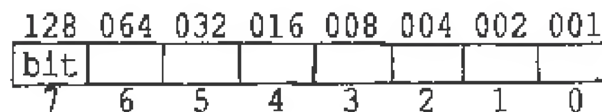
8-bits Z80-register, dat meer mogelijkheden heeft dan de andere registers.
De meeste rekenkundige en logische bewerkingen verlopen via dit register (dat daarom ook wel reken-register wordt genoemd).

### Vlag-register of F-register

8-bits Z80-register, waarvan de afzonderlijke bits beïnvloed worden door rekenkundige en logische bewerkingen.

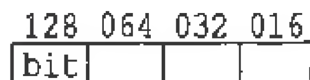
## *1.5 Bytes en bits*

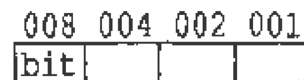
```
------------------------------------------------------------
BYTE - NIBBLE - BIT
------------------------------------------------------------
1  BYTE: groep van 8 bits           BIT = BInary digiT
   BIT = BInary digiT, kleinste eenheid waarmee de Z80-
   processor werkt. De waarde van een bit is 0 of 1.
```




```
------------------------------------------------------------
2  NIBBLE, groep van 4 bits

   high nibble, bits 7 - 4       low nibble, bits 3 - 0
```




```
------------------------------------------------------------
```

```
Een hexadecimaal getal verwijst naar de nibbles van een
byte

voorbeeld: &H E1                E = high nibble 1110 (14)
                                1 = low nibble  0001  (1)

--------------------------------------------------------
De nibble wordt als rekeneenheid gebruikt bij het
rekenen in BCD-notatie (Binary Code Decimal).
Elke nibble bevat dan een decimaal getal (0 - 9)
--------------------------------------------------------
```

*figuur 1.5a*

Het geheugen van de computer is opgebouwd uit bytes: opslagruimten, waarin getalswaarden worden geplaatst. Een byte kan een waarde bezitten die varieert van 0 tot en met 255. In Basic worden afzonderlijke bytes over het algemeen niet direkt aangesproken. Er zijn echter een aantal instrukties, die een direkt aanspreken mogelijk maken:

- de instrukties POKE, PEEK, VPOKE en VPEEK; hiermee kunnen we afzonderlijke bytes in het RAM en het video-RAM uitlezen of beïnvloeden. Zo zal het kommando POKE 50000,300 een foutmelding opleveren; hetgeen bewijst dat een byte geen waarden groter dan 255 mag bevatten.

- de logische funkties AND, OR en XOR; hiermee kunnen we de opbouw van een geheugen-byte onderzoeken en indien gewenst beïnvloeden.
  Bijvoorbeeld IF (PEEK(50000) AND 0) = 0 THEN PRINT "JUIST"

In machinetaal (juister gezegd: assembler-taal) kennen we deze instrukties ook; vandaar dat er in dit boek een test-programma is opgenomen, waarin logische bewerkingen worden uitgevoerd met getallen (hoofdstuk 6.2).
Het is een Basic-programma dat de bytes in binaire vorm op het scherm plaatst (zie figuur 1.5a).

Voordat we overgaan tot het uittesten van instrukties moeten we eerst exakt bepalen wat een byte is en wat we ermee kunnen doen.

Dat is noodzakelijk omdat het programmeren in machinetaal zich afspeelt op het elementaire niveau van bytes en bits.

Zoals we kunnen zien in figuur 1.5a is een byte opgebouwd uit 8 eenheden, de bits. De bit is de kleinste eenheid waarmee de Z80-processor werkt. Een bit kunnen we zien als een simpel schakelelement, omdat de inhoud van een bit ofwel 0 ofwel 1 is.

Als de inhoud van een bit 1 is zeggen we dat de bit 'hoog' of aan of 'gezet' is. Is de inhoud van een bit 0 dan zeggen we dat hij 'laag' is of uit of niet gezet (in het Engels 'reset'). Er zijn dus maar twee standen mogelijk: AAN of UIT. De term 'binair' verwijst naar die twee standen. Binair betekent twee-voudig of twee-tallig. Een binair rekensysteem is een systeem dat met twee getallen werkt (de getallen 0 en 1).

Iedere bit in een byte heeft behalve een binaire waarde een decimale waarde. Die decimale waarden staan in figuur 1.5a afgedrukt boven de bokjes die de bits voorstellen. Is een bepaalde bit hoog (dus gelijk aan 1) dan wordt aan de getalswaarde van de byte het getal toegevoegd dat bij het betreffende bit behoort. Maken we het bit 0 dan wordt de getalswaarde van de byte verminderd met de decimale waarde die bij dat bit hoort.

De decimale waarde die bij een bit hoort wordt op de volgende wijze bepaald:

We gaan uit van het grondgetal 2 en we voeren daarmee een machtsverheffing uit. Willen we bijvoorbeeld weten welke decimale waarde bij bit 3 behoort, dan verheffen we het getal 2 tot de macht 3. Het getal is dus: 2*2*2 = 8

In figuur 1.5b worden de meest gebruikte rekensystemen onder elkaar gezet. We zien dat alle systemen uitgaan van hetzelfde principe: een grondgetal tot een bepaalde macht verheffen (waarbij geldt dat een getal dat tot de macht nul wordt verheven altijd gelijk wordt aan één).

```
------------------------------------------------------------
REKENSYSTEMEN
die bij het machinetaalprogrammeren worden gebruikt:
------------------------------------------------------------
1  DECIMAAL                            grondtallen:
   rekenen in machten van 10
                                       0,1,2,3,4,5,6,7,8,9
   10 +10 +10 +10 + . . . +10          decimaal: tientallig

voorbeeld: het getal 225               2*10 +2*10 +5*10  =
                                       200+20+5
------------------------------------------------------------
2  BINAIR                              grondtallen:
   Rekenen in machten van 2
                                       0 en 1
   2 +2 +2 +2 + . . . +2               binair: tweetallig

voorbeeld: 225 = &B 1110 0001          1*2 +1*2 =1*2 +0*2 +
                         (14/01)       0*2 +0*2 +0*2 +1*2  =
                                       128+64+32+0+0+0+1
------------------------------------------------------------
3  HEXADECIMAAL                        grondtallen:
   Rekenen in machten van I6
                                       0,1,2,3,4,5,6,7,8,9,A,
   16 +16 +16 +16 + . . . +16          B,C,D,E,F

   A=10 B=11 C=12 D=13 E=14 F=15       hexadecimaal:16-tallig

voorbeeld: 225 = &H E1                 E*16 +1*16  =
                 (14/01)               14*16+1 = 224+1
------------------------------------------------------------
```

*figuur 1.5b*

De computer schakelt voortdurend bits aan en uit en om zijn gedrag goed te kunnen begrijpen is het nuttig om ons zelf te verplaatsen in de positie van de computer; d.w.z. we gaan zelf schakelen met bits.

De Z80-microprocessor kent een aantal instrukties die een direkte beïnvloeding en kontrole van afzonderlijke bits mogelijk maken. Hoewel we pas in het volgende hoofdstuk overgaan tot het bespreken van de Z80-instrukties willen we daar hier al op vooruitlopen

door te wijzen op een drietal bit-manipulatie-instrukties:

- BIT bitnummer,register:
  kijk of een bit van een register 0 of 1 is;

- SET bitnummer,register:
  maak het aangegeven bit van het register hoog (dus gelijk aan 1);

- RES bitnummer,register:
  maak het aangegeven bit van het register laag (dus gelijk aan 0).

Het verdient aanbeveling het uitvoerige Basic-testprogramma, dat in hoofdstuk 6.3 is opgenomen, in te toetsen en uit te proberen. Het voordeel van een dergelijk Basic-programma is dat we kunnen zien wat we doen.

Werken in machinetaal is veelal een 'abstrakte bezigheid', d.w.z. we werken met getallen en instrukties, die ons niets konkreets vertellen over de opbouw van een byte - we zien het resultaat van de beïnvloeding niet. Het Basic-testprogramma echter laat na iedere instruktie de inhoud zien van de byte en in het geval van de bit-test-instruktie vertelt het ons tegelijkertijd op welke wijze het vlag-register wordt beïnvloed.

In dit hoofdstuk beperken we ons tot een enkel voorbeeld:

We willen in machinetaal programmeren en we plaatsen om te beginnen de waarde 0 in het rekenregister (de akku). Alle bits van dit register zijn nu laag (dus 0).

Willen we bit 7 inschakelen dan kunnen we dat doen met behulp van de Z80-instruktie: 'SET 7,A'.

Willen we het bit uitschakelen dan gebruiken we de instruktie: 'RES 7,A'.

Willen we kijken of een bit 0 of 1 is dan gebruiken we de instruktie: 'BIT 7,A'.

In het laatste geval plaatst de processor het resultaat van zijn onderzoek in het vlag-register (zie figuur 1.5c): is het bit ingeschakeld (1)

dan wordt de zero-vlag 0 gemaakt; is het bit uitgeschakeld dan wordt de zero-vlag 1 gemaakt.

---

F- OF VLAG-REGISTER

---

| sign | zero | ---- | ---- | ---- | P/V | ---- | carry | |
|---|---|---|---|---|---|---|---|---|
| 1=M<br>0=P | 1=Z<br>0=NZ | | | | 1=PE<br>0=PO | | 1=C<br>0=NC | |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | bitnummer |

---

Sign = tekenvlag (M = minus/negatief; P = positief)

---

Zero-vlag geeft aan of het resultaat van de vergelijkingen (logisch of rekenkundig) gelijk is aan 0 (Z = 1) of ongelijk aan 0 (Z=0)

---

P/V-vlag wordt gebruikt als pariteitsvlag bij logische vergelijkingen; in dat geval wijst PE op 'even pariteit' (het getal bevat een even aantal enen) en PO op 'oneven pariteit'.
Bij rekenkundige bewerkingen wordt de P/V-vlag gebruikt als overflow-vlag (bij tweekomplement-berekeningen). PE wijst in dit geval op overflow.
Bij blok-zoek-en-verplaats-instrukties wordt de P/V-vlag gebruikt om aan te geven of BC gelijk is aan 0 of niet: P/V=0 (PO) als BC=0

---

De carry-vlag wordt 1 als het resultaat van een rekenkundige bewerking niet in het gebruikte register (-paar) past; als bij vergelijkingen het af te trekken getal groter is dan het te bewerken getal; als bij roteer- en schuifbewerkingen het uitgeschoven bit gelijk is aan 1

---

*figuur 1.5c*

Met behulp van het Basic-testprogramma kan ook de vlagbeïnvloeding door de bit-instruktie worden bestudeerd. Voordat men verdergaat met lezen verdient het aanbeveling via dit program-

ma de relatie bit-zero-vlag goed te bekijken. Vlag-beïnvloeding is een essentieel gegeven en kennis ervan is absoluut noodzakelijk wanneer we machinetaal-programma's willen schrijven.

Voor diegenen die met behulp van de assembler deze instrukties willen testen geven we hieronder een eenvoudig programmaatje:

```
10 ' ORG 49500
20 ' LD A,0 ;maak inhoud akku gelijk aan 0
30 ' SET A,7 ;maak bit 7 hoog
40 ' LD (50000),A ;plaats inhoud A in lokatie 50000
50 ' RET ;keer terug naar Basic
60 ' END
```

Wanneer met behulp van de Basic-PEEK-funktie geheugenlokatie 50000 wordt uitgelezen zal (als alles goed is verlopen) op het scherm de waarde 128 worden afgedrukt.
Wanneer we kijken naar figuur 1.5a kunnen we konstateren dat de waarde 128 inderdaad behoort bij bit 7.

De waarde van een byte zegt ons dus iets over de positie van de bits. We kunnen dat zelf konstateren door met behulp van de Basic-funktie BIN$ de binaire waarde van een getal op het scherm af te drukken:

PRINT BIN$(255) geeft het resultaat: 11111111
PRINT BIN$(128) geeft het resultaat: 10000000

Kleinere waarden worden niet korrekt weergegeven, omdat de hoge bits die 0 zijn niet worden afgebeeld. Zo geeft de opdracht PRINT BIN$(0) het resultaat '0'. In de test-programma's wordt deze 'fout' gekorrigeerd.

## *1.6 Lijst van gebruikte begrippen*

### Byte

Groep van 8 bits, die elk een decimale waarde vertegenwoordigen. Zijn alle bits laag (0) dan is de decimale getalswaarde van de byte 0;

zijn alle bits hoog (1) dan is de waarde 255. De getallen kunnen zowel een rekenkundige- als een symbolische betekenis hebben.

**Bit (binary digit)**

Kleinste eenheid waarmee de Z80-processor werkt. Een bit kent slechts twee standen: 1 (aan - hoog - gezet) of 0 (uit - laag - niet-gezet/'reset').

**Binair**

Tweetallig - werken met twee getallen: 0 en 1. Binair rekensysteem: systeem dat rekent met machten van 2. (Zie figuur 1.5b.)

**Logische bewerking**

Bewerking waarbij de bits van een byte een voor een met elkaar worden vergeleken. Aan het einde van elke vergelijking wordt, afhankelijk van de aard van de logische bewerking (AND, OR, XOR, CP, CPL of NEG), een nieuw resultaat in het rekenregister geplaatst. (Zie het Basic-testprogramma in hoofdstuk 6.2.)

**Bit-manipulatie-instrukties**

Z80-instrukties, waarmee afzonderlijke bits in een byte getest en beïnvloed kunnen worden (BIT - SET - RES). (Zie het Basic-testprogramma in hoofdstuk 6.3.)

## *1.7 Logische bewerkingen*

Zowel de Basic-programmeertaal als de assembler-programmeertaal kennen instrukties waarmee 'logische bewerkingen' kunnen worden uitgevoerd.
In het Basic-testprogramma, waarnaar reeds verschillende malen werd verwezen, worden de logische funkties AND, OR en XOR getest.

In het kort willen we in dit deel-hoofdstuk aangeven welk effekt de verschillende logische bewerkingen hebben: we moeten ons daarbij realiseren dat bij een logische bewerking de bits afzonderlijk wor-

den vergeleken:

Bit 0 van het 1e byte wordt vergeleken met bit 0 van het 2e byte.

Bit 1 van het 1e byte wordt vergeleken met bit 1 van het 2e byte.

Enzovoort...
Het resultaat van de vergelijking wordt in het rekenregister van de Z80-processor geplaatst.

**a. De AND-bewerking**

Deze bewerking geeft als resultaat 1, wanneer de twee bits die met elkaar worden vergeleken allebei hoog (1) zijn.

Dus: 1 AND 1 = 1

Is een van beide bits laag (0) dan zal het resultaat altijd 0 zijn.

Dus: 0 AND 1 = 0
1 AND 0 = 0
0 AND 0 = 0

Met behulp van de AND-instruktie kunnen we bits in een byte uitzetten (0 maken). We moeten in dat geval de betreffende byte vergelijken met een byte, waarin de bits die we uit willen schakelen laag (0) zijn (we spreken ook wel over 'maskeren' of het uitvoeren van een 'masker-operatie').
Dus: willen we bits 4 tot 7 uitschakelen dan moeten we een vergelijking uitvoeren met een getal dat de volgende binaire vorm heeft: 00001111
Wat ook de inhoud mag zijn van het eerste byte, de logische AND-bewerking zal altijd de bits 4 - 7 in de uit-stand zetten. We kunnen dit kontroleren met behulp van het Basic-testprogramma.

**b. De OR-bewerking**

Deze bewerking geeft als resultaat 1, wanneer een van de twee bits die met elkaar worden vergeleken 1 is.

Dus: 1 OR 0 = 1

0 OR 1 = 1
1 OR 1 = 1

Het resultaat van een OR-bewerking is alleen dan 0, wanneer beide bits 0 zijn. Dit laatste feit moeten we goed onthouden, omdat we dit gegeven kunnen gebruiken bij het ontwerpen van programmalussen.

Met behulp van de OR-instruktie kunnen we de bits in een byte aanzetten (1 maken).
We moeten in dat geval de betreffende byte vergelijken met een byte, waarin de bits die we aan willen zetten hoog (1) zijn. Dus willen we bits 4 - 7 aanzetten, dan moeten we een vergelijking uitvoeren met een getal dat de volgende binaire vorm heeft: 11110000
De inhoud van de eerste vier bits (0 - 3) zal niet worden gewijzigd: een bit dat 1 is blijft 1 - een bit dat 0 is blijft 0. Kontrole is mogelijk met behulp van het Basic-testprogramma.

**c. De XOR-bewerking**

Deze bewerking geeft als resultaat 1, wanneer de twee bits die met elkaar worden vergeleken niet aan elkaar gelijk zijn:

Dus: 1 XOR 0 = 1
0 XOR 1 = 1

Het resultaat van een XOR-bewerking is 0, wanneer de de beide bits gelijk zijn aan elkaar.

Dus: 1 XOR 1 = 0
0 XOR 0 = 0

Met behulp van de XOR-instruktie kan (bijvoorbeeld) gekeken worden of de inhouden van twee bytes aan elkaar gelijk zijn. Een vergelijking zal het resultaat 0 opleveren wanneer de inhouden gelijk zijn aan elkaar.
Dus: (A XOR B) = 0 als A=B.
Kontrole is mogelijk met behulp van het Basic testprogramma.

De Z80-instruktieset kent nog een aantal logische bewerkingen: *ComPare*, *ComPLement* en NEG (negatief maken).
Deze instrukties hebben echter geen Basic-ekwivalent, zodat ze in

het hoofdstuk, waarin de Z80-instrukties aan de orde komen, zullen worden besproken.

## *1.8 Lijst van gebruikte begrippen*

**Masker**

Vergelijkingsbyte waarvan een aantal bits, afhankelijk van de logische bewerking die wordt uitgevoerd, hoog of laag zijn gemaakt, teneinde de bits in het doel-byte te beïnvloeden.

**Z80-instruktieset**

Geheel van instrukties, waarmee de Z80-microprocessor geprogrammeerd kan worden. Deze instrukties hebben een binaire vorm, omdat de processor alleen kan werken met binaire getallen. Om het programmeren in machinetaal wat gemakkelijker te maken werd een taal ontworpen waarin de binaire instrukties werden omgezet in lettertekens ('mnemonics'): de assemblertaal.

Wanneer we in dit boek spreken over machinetaal dan verwijzen we in feite naar deze assemblertaal. De assembler zet de assemblertaal om in getallen, die door de Z80-processor kunnen worden geïnterpreteerd.

## *1.9 Nibbles en hexadecimale getallen*

Wanneer we kijken naar figuur 1.5a dan zien we dat een groepje van vier bits wordt aangeduid met het begrip 'nibble'. De bits 0 - 3 vormen de lage 'nibble'; de bits 4 - 7 vormen de hoge 'nibble'.

Het gebruik van 'nibbles' is vooral van belang wanneer we willen rekenen met behulp van de BCD-(binary-coded-decimal)-notatie. In dit boek zullen we weinig aandacht besteden aan deze rekenwijze, omdat het schrijven van relatief eenvoudige rekenprogramma's een uiterst ingewikkelde zaak is. De Basic-interpreter maakt gebruik van deze rekenwijze, wanneer er wordt gewerkt met variabelen die in enkele of dubbele precisie staan.
In hoofdstuk 4 besteden we aandacht aan een aantal Basic-ROM-

routines, waarmee we berekeningen kunnen uitvoeren. Via deze routines maken we op indirekte wijze gebruik van de BCD-rekenmethode.

Hier willen we in de eerste plaats de aandacht vestigen op het verband dat er bestaat tussen een hexadecimaal getal en de 'nibbles' van een byte.

Laten we als voorbeeld het getal 255 nemen.

De binaire vorm van dit getal is: 11111111 De byte bestaat hier dus uit twee groepjes van bits, die allemaal hoog (1) zijn.
Willen we de hexadecimale notatie van het getal 255 achterhalen dan moeten we de twee nibbles los van elkaar beschouwen. Dat betekent dat in dit geval de inhoud van elke nibble 15 is. Dit decimale getal wordt omgezet in een hexadecimaal getal (15 = F), zodat de totale inhoud F + F is: Dat is hexadecimaal FF (zie ook figuur 1.5b).

Het gebruik van de hexadecimale notatie kan dus erg nuttig zijn, omdat het hexadecimale getal ons (in tegenstelling tot het decimale getal) iets kan vertellen over de bit-opbouw van een byte. Het is niet zo dat het gebruik van hexadecimale getallen een absolute plicht is bij het schrijven van machinetaal-programma's; die verkeerde indruk wordt nogal eens gewekt door sommige auteurs. De hexadecimale notatie gebruiken we alleen daar waar het nuttig en nodig is. In de andere gevallen kunnen we rustig de decimale schrijfwijze hanteren!

We besluiten dit korte hoofdstuk met een tweede voorbeeld.

Stel we hebben het hexadecimale getal: DE.

Dit getal verwijst naar de twee nibbles van een byte: de hoge nibble heeft de inhoud D; de lage nibble heeft de inhoud E. De hexadecimale grondgetallen lopen van 0 tot 15, waarbij de letters A tot F de getallen 10 tot 15 symboliseren: A=10, B=11, C=12, D=13, E=14...
De inhoud van de hoge nibble is dus 13; de inhoud van de lage nibble is 14.

Wanneer we een masker-bewerking willen uitvoeren, dan kan het

gebruik van de hexadecimale notatie erg handig zijn. Stel we willen een AND-bewerking uitvoeren, waarbij de bits 4 tot 7 uitgeschakeld moeten worden. De inhoud van de hoge nibble moet in dat geval 0 zijn; de inhoud van de lage nibble 15 (zie hoofdstuk 1.7 a). De AND-bewerking zal dus uitgevoerd moeten worden met het hexadecimale getal: 0F.

## *1.10 Lijst van gebruikte begrippen*

### Nibble

Groepje van 4 bits; wordt als rekeneenheid gebruikt bij het rekenen in BCD-notatie.

### BCD (Binary Coded Decimal) rekenen

Decimaal rekenen met behulp van de binaire kode. Elk decimaal getal (0 - 9) wordt opgeslagen in een groepje van 4 bits (de 'nibble').
Een byte vormt op die manier een decimaal getal van 2 cijfers. Voorbeeld: de binaire vorm van het BCD-getal 11 is: 0001 0001 De decimale waarde van dit binaire getal is: 16 + 1 = 17
Het grootste decimale getal dat volgens de BCD-methode opgeslagen kan worden in een byte is het getal 99: binair 1001 1001

### Hexadecimaal rekensysteem

Rekensysteem dat werkt met zestien grondgetallen, waarbij de getallen 10 tot en met 15 worden vertegenwoordigd door de letters A, B, C, D, E en F. Een hexadecimaal getal verwijst naar de nibble's van een byte (zie ook figuur 1.5b).

# 2. De Z80 microprocessor

## *2.1 De Z80-instruktieset*

De instrukties die ons in staat stellen de Z80-processor direkt aan te spreken kunnen onderverdeeld worden in een aantal hoofdkategorieën.

*1 De load (= laad)- en exchange (= verwissel)-instrukties:*
deze instrukties hebben betrekking op de 8-bits-registers (A, B, C, D, E, H en L) en de 16-bits-registers of registerparen (SP, IX, IY, BC, DE en HL).

*2 Eenvoudige rekenkundige instrukties:*
optellen (ADD, ADC, INC) en aftrekken (SUB, SBC, DEC); deze instrukties werken zowel met 8-bits-registers als met 16-bits-registers (of registerparen).

*3 Logische instrukties:*
AND, OR, XOR, ComPare, ComPLement, NEGative; deze instrukties hebben betrekking op de inhoud van de akkumulator (een 8-bits-rekenregister).

*4 Roteer- en schuif-instrukties:*
a. Roteer-instrukties:
   8-bits-rotatie; RL, RLA, RR, RRA;
   9-bits-rotatie; RLC, RLCA, RRC, RRCA.
b. Schuif-instrukties:
   SLA, SRA, SRL (8-bits-bewerkingen).

*5 Bit-manipulatie-instrukties:*
a. Test-instruktie:
   BIT bitnummer,register.
b. Schakel-instrukties:

SET bitnummer,register (bit 1 maken);
RES bitnummer,register (bit 0 maken).

Al deze instrukties werken met eenheden van 8 bits.

*6 JumP (=sprong)- en CALL/ReSTart (subroutine)-instrukties:*

a. Sprong-instrukties:
   Direkte sprong - JumP:
   Spring naar een adres
   Relatieve sprong - Jump Relative:
   spring *n* bytes vooruit of terug.

b. Subroutine-instrukties:

CALL adres (elk adres geoorloofd);
RST adres (beperkt aantal adressen).
Deze instrukties hebben (direkt of indirekt) betrekking op de adressen 0 - 65535.

*7 Stack-(= stapel)instrukties:*

a. startadres veranderen: LD SP,nn;
   inhoud uitwisselen met registers HL,IX en IY: EX (SP),rr.

b. PUSH- en POP-instrukties:
   PUSH - op de stapel leggen (in het geheugen plaatsen);
   POP - van de stapel halen (uit het geheugen halen).

Deze instrukties hebben betrekking op 16-bits-registers of registerparen (AF, BC, DE, HL, IX en IY).

*8 Blok-zoek- en blok-verplaats-instrukties:*

a. zoek-instrukties: CPI, CPIR, CPD, CPDR;
b. verplaats-instrukties: LDI, LDIR, LDD, LDDR.

Deze instrukties hebben betrekking op de inhoud van de geheugenlokaties 0 tot 65535.

*9 Input (=invoer)- en Output (=uitvoer)-instrukties:*

deze instrukties maken een direkte invoer en uitvoer van gegevens van en naar randapparatuur (scherm, printer, e.d.) mogelijk.

*10 Rest-instrukties,*

die niet binnen één van de hoofd-kategorieën vallen, zoals NOP (no operation), HALT (wacht op interrupt) en de interrupt-instrukties EI (enable interrupt - maak interrupts mogelijk) en DI (disable interrupt - maak interrupts onmogelijk).

## *2.2 Lijst van gebruikte begrippen*

**8-bits-registers**

Z80-registers, die de decimale getallen 0 - 255 kunnen bevatten: A, B, C, D, E, H, L, (zie figuur 1.3c). Er zijn twee bijzondere 8-bits-registers:

a. de akku/akkumulator of het A-register - dit is een rekenregister;

b. het vlag- of F-register - hierin wordt na bepaalde bewerkingen informatie opgeslagen, die op een indirekte wijze door ons kan worden uitgelezen (zie figuur 1.5c).

**16-bits-registers of register-paren**

Z80-registers, die de decimale getallen 0 - 65535 kunnen bevatten: SP, IX, IY, BC, DE, HL (zie figuur 1.3c).
De registers BC, DE en HL worden 'register-paren' genoemd; ze zijn opgebouwd uit twee 8-bits-registers:

B en C vormen samen BC (B = hoge helft - C = lage helft);
D en E vormen samen DE (D = hoge helft - E = lage helft);
H en L vormen samen HL (H = hoge helft - L = lage helft).

De registers SP, IX en IY worden 16-bits-registers genoemd, omdat de afzonderlijke helften ervan niet als 8-bits-register gebruikt kunnen worden.
Helemaal korrekt is dat niet, omdat de IX- en IY-registers (ook wel index-registers genoemd) wel degelijk opgesplitst kunnen worden in twee helften, die elk afzonderlijk beinvloed kunnen worden. (De bezitters van het assembler/disassembler-pakket FLASH vinden in de handleiding meer informatie hierover.)

**Interrupt**

Een groot aantal malen per sekonde (50 maal) voert de Z80-processor automatisch een aantal opdrachten uit. Een Basic-programma of een zelfgeschreven machinetaal-programma wordt op die momenten onderbroken en de computer springt naar een standaard-ROM-routine (KEYINT), die zich bevindt op adres

&H0038. Deze standaard-routine voert een aantal handelingen uit, waarvan de belangrijkste de aftasting van het toetsenbord is (er wordt gekeken of er toetsen werden ingedrukt en zoja welke toetsen).
Op die manier is het mogelijk dat een Basic-programma tijdens de uitvoering beëindigd kan worden door het indrukken van de toetsen Ctrl en Stop. Ook een machinetaal-programma kan dankzij deze interrupt-mogelijkheid beëindigd worden:
Voorwaarde is dat direkt na het opstarten van de machine het getal 1 in de systeem-variabele ENSTOP (&HFBB0) wordt geplaatst. Wanneer dat is gedaan zal vanuit elk programma (Basic- en machinetaal) teruggesprongen worden naar de (direkte) Basic-mode na het tegelijkertijd indrukken van de toetsen CTRL, SHIFT, GRAPH en CODE.
(Het spreekt voor zichzelf dat het onmogelijk maken van de interrupts met hehulp van de instruktie DI - disable interrupt - een terug keer naar de Basic-mode zal verhinderen.)

In Basic-programma's kunnen we van de Interrupt-mogelijkheid van de Z80-processor gebruik maken via de instruktie INTERVAL.

## *2.3 De load-instrukties*

```
--------------------------------------------------------------
LD r,r          r = 8-bits-register (A, B, C, D, E, H, L)
LD r,n          n = getal (0 - 255)
--------------------------------------------------------------
LD r,(rr)       rr = 16-bits-register (HL, IX+d, IY+d)
LD (rr),r       () haakjes geven aan dat het gaat om de
LD (rr),n       inhoud van de geheugenplaatsen, waarnaar
                rr verwijst.
                d = displacement of verplaatsing;
                positief: 0 t/m 127
                negatief: 1 t/m 128
--------------------------------------------------------------
LD A,(m)        m = 16-bits-register (BC, DE, HL, IX+d,
LD (m),A        IY+d) of getal (0-65535); A = akkumulator
--------------------------------------------------------------
LD rr,nn        rr = 16-bits-register (BC, DE, HL, SP, IX,
LD rr,(nn)      IY)
```

```
LD (nn),rr      nn=getal (0 - 65535)
                () haakjes geven aan dat het gaat om de
                inhoud van de geheugenplaatsen waarnaar
                nn wijst.
------------------------------------------------------------
LOAD-instrukties beinvloeden het F- of vlag-register niet
------------------------------------------------------------
```

*figuur 2.3a*

Zoals we in Basic-programma's variabelen een bepaalde inhoud kunnen geven met behulp van de opdracht LET (b.v. LET A=12), zo kunnen we met behulp van de Z80-laad-instrukties een register, een registerpaar of een geheugenplaats een bepaalde inhoud geven. Zo zal de instruktie LD A,12 het getal 12 in het A-register plaatsen. De instruktie LD (50000),A plaatst de inhoud van A in het geheugen (in Basic zouden we het kommando POKE 50000,12 gebruiken). De instruktie LD HL,1200 plaatst het getal 1200 in het registerpaar HL.

Getallen die groter zijn dan 255 worden ontbonden in twee 8-bits-getallen: de hoge byte, die vermenigvuldigd moet worden met 256 om de juiste waarde te achterhalen, en de lage byte, die 'onveranderd' blijft.

We kunnen dit kontroleren door de inhoud van register H in de akku (het A-register) te plaatsen en vervolgens die inhoud in een bepaalde geheugenlokatie te plaatsen, die we met behulp van de Basic-PEEK-funktie kunnen uitlezen.
Voorbeeld:

```
10 ' ORG 49500
20 ' LD HL,1200
30 ' LD A,H ;plaats inhoud H in de akku
40 ' LD (50000),A ;plaats inhoud A in lokatie 50000
50 ' RET ;return - keer terug naar Basic
60 ' EIND END
```

In regel 20 wordt de inhoud van register H in register A geplaatst, waarna in regel 30 de inhoud van A in het geheugen wordt gezet. Deze procedure is noodzakelijk omdat het niet mogelijk is de inhoud van de 8-bits-registers B, C, D, E, H en L rechtstreeks in het geheugen te plaatsen.

Wel bestaat er de mogelijkheid om de inhoud van een register-paar direkt in het geheugen te plaatsen [zie in figuur 2.3a de instruktie LD (nn),rr].
In dat geval worden twee geheugen-bytes gereserveerd: de lage helft wordt in de eerste byte gezet; de hoge helft in het tweede byte.

Wanneer we met behulp van de assembler de routine in het geheugen hebben geplaatst en de uitvoering ervan tot stand hebben gebracht met behulp van de Basic-instrukties DEFUSR en USR [in dit geval: DEFUSR0=49500:Z=USR0(0)] kunnen we het resultaat uitlezen door de opdracht PRINT PEEK(50000) in te toetsen.
We zullen zien dat dat het getal 4 op het scherm verschijnt. Dat betekent dat de hoge helft van registerpaar HL (register H) de waarde 4 bevat. Vermenigvuldiging van dit getal met de waarde 256 levert het getal 1004 op. Register L (de lage helft van HL) zal daarom de waarde 196 (1200 minus 1004) moeten bevatten.

Om dit te kontroleren ontwerpen we een routine die gebruik maakt van de mogelijkheid om de inhoud van de 8-bits-registers H en L tegelijker tijd in het geheugen te plaatsen.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD HL,1200
30 ' LD (50000),HL
40 ' RET ;return - terug naar Basic
50 ' EIND END
```

Wanneer we deze routine uitvoeren zal de inhoud van registerpaar HL in de geheugenlokaties 50000 en 50001 worden gezet.
Lokatie 50000 bevat het lage deel (inhoud register L).
Lokatie 50001 bevat het hoge deel (inhoud register H).
We kunnen dit kontroleren door beide lokaties uit te lezen met behulp van de Basic-funktie PEEK. De getallen 196 en 4 zullen op het scherm moeten verschijnen.

We zien dus dat we vanuit het machinetaal-programma informatie op een vaste plek in het geheugen kunnen plaatsen. Die informatie kan zowel vanuit machinetaal-programma's als vanuit Basic-programma's benaderd worden. We hebben in feite een 'variabele'

ontworpen. Die variabele kan uitgelezen worden wanneer het *adres* ervan wordt opgegeven (of nadat binnen een assembler-bron-programma het adres ervan verbonden is met een *label* - het Label fungeert in dat geval als variabelennaam).

Het plaatsen van informatie in het geheugen kan op twee manieren gebeuren:

1. Op direkte wijze

Voorbeeld: LD (50000),A

De informatie die ligt opgeslagen in register A wordt direkt in geheugen-lokatie 50000 geplaatst;

2. Op indirekte wijze

Voorbeeld:
```
LD HL,50000
LD (HL),A
```

Het adres van de geheugen-lokatie wordt eerst in registerpaar HL opgeslagen en daarna wordt de informatie die in A aanwezig is doorgegeven naar de betreffende geheugen-byte.

Het voordeel van indirekte verplaatsing is dat bij het opvullen van meerdere geheugen-bytes met gegevens niet steeds opnieuw het adres opgegeven hoeft te worden.

Dus niet:
```
LD (50000),A
LD A,B
LD (50001),A
enz.
```

maar:
```
LD HL,50000
LD (HL),A
INC HL ;HL=HL+1
LD (HL),B
enz.
```

In figuur 2.3a zien we dat indirekte verplaatsing van informatie ook mogelijk is met behulp van de indexregisters IX en IY. Het gebruik

van deze registers vereist het toevoegen van een verplaatsingswaarde (0 tot 127).

Voorbeeld: LD IX,50000
LD (IX+0),A
LD (IX+1),B

Geheugenlokatie 50000 wordt gevuld met de inhoud van register A; lokatie 50000 plus 1 wordt gevuld met de inhoud van register B. Ook negatieve getallen (-1 tot -128) mogen als verplaatsingswaarde worden gebruikt.

Voorbeeld: LD IX,50002
LD (IX-2),A
LD (IX-1),B

Het resultaat zal hetzelfde zijn: Inhoud A wordt geplaatst in lokatie 50002 min 2; inhoud B in lokatie 50002 min 1 (50000 en 50001). Index-registers zijn vanwege deze mogelijkheden bij uitstek geschikt om een variabelen-gebied te kreeren in het RAM-geheugen.

## *2.4 Lijst van gebruikte begrippen*

### DEFUSR (define user function)

Basic-kommando, met behulp waarvan het startadres van een machinetaal-routine doorgegeven kan worden aan de computer. Men kan in totaal 10 startadressen vastleggen (0 - 9); ze worden in het werkgeheugen van de computer opgeslagen vanaf het adres &HF39A (USRTAB).
Het kommando wordt op de volgende manier gebruikt:

DEFUSR x = adres
x = 0 - 9 (een van de tien mogelijkheden).

### USR (user function)

Basic-funktie, met behulp waarvan een machinetaal-routine in werking kan worden gesteld. Deze funktie maakt gebruik van de startadressen die met behulp van het DEFUSR-kommando in het werk-

gebeugen werden geplaatst; ze kan daarom alleen worden gebruikt wanneer een definiëring van startadressen al heeft plaatsgevonden! De funktie wordt op de volgende manier gebruikt:

Variabele = USR x (p)
x = 0 - 9 (een van de 10 startadressen)
p = parameter (0 wanneer men geen informatie wil doorgeven aan de machinetaal-routine).

## *2.5 De verwissel-instrukties*

Zoals het in Basic mogelijk is de inhoud van twee variabelen via de instruktie SWAP om te wisselen, zo kan bij het programmeren in machinetaal de inhoud van 16-bits-registers (of registerparen) worden verwisseld met behulp van de instruktie *EX*change.

De Z80-instruktie-set kent de volgende verwissel-instrukties:

a EXX
Deze instruktie plaatst de inhouden van de registers BC, DE en HL in de hulp-registers BC', DE' en HL' (zie figuur 1.3c) en omgekeerd.
Men kan met behulp van deze instruktie tijdelijk de inhoud van de betreffende registers bewaren. Men moet zich goed realiseren dat het gebruik van deze instruktie betekent dat de inhouden van de registers B, C, D, E, H en L veranderen!

b EX AF,AF'
Deze instruktie ruilt de inhouden van de registers A+F en A'+F' om.

c EX DE,HL
Verwisselt de inhouden van de registerparen DE en HL.

d EX (SP),rr (rr = HL, IX of IY)
Deze instruktie verwisselt de inhouden van het stapel-register en het registerpaar dat wordt aangegeven door rr. (De beginnende programmeur zal deze instrukties zelden gebruiken.)

### *2.6 Lijst van gebruikte begrippen*

## SWAP

Basic-kommando, waarmee de waarden van twee string-variabelen of twee numerieke variabelen kunnen worden omgewisseld.

## Stapel

Een gebied in het RAM-geheugen, waarin door de Z80-processor tijdelijk de inhouden van register-paren worden opgeslagen. Het startadres van de stapel kan men achterhalen door het uitlezen van de systeem-variabele &HF674 (STKTOP):

```
PRINT PEEK(&HF674) + 256*PEEK(&HF675)
```

## 2.7 De rekenkundige of aritmetische instrukties

```
-------------------------------------------------------------------
1  Instrukties waarbij de akku centraal staat   |
                                                 |
ADD A,                  r  = A, B, C, D, E, H, L | C = Carry
ADC A,  \  r, n         n  = getal (0 - 255)     | Z = Zero
SUB     / of (rr)       rr = HL, IX+d, IY+d      | S = Sign
SBC A,                                           |   of teken
                                                 | V = over-
                                                 |     flow
C-, Z-, S- en V-vlaggen worden beinvloed         |
-------------------------------------------------------------------
2  Instrukties rond de registers HL, IX en IY    |
                                                 |
ADD HL,rr                rr = HL, DE, BC, SP     |
ADD IX,rr                rr = IX, DE, BC, SP     |
ADD IX,rr                rr = IY, DE, BC, SP     |
                                                 |
Alleen de carry-vlag wordt beinvloed             |
                                                 |
ADC HL  \  rr            rr = BC, DE, HL, SP     |
SBC HL, /                rr = BC, DE, HL, SP     |
                                                 |
C-, Z-, S- en V-vlaggen worden beinvloed         |
-------------------------------------------------------------------
3  Algemeen:                                     |
                                                 |
INC   \   r of           r  = A, B, C, D, E, H, L|
DEC   /   (rr)           rr = Hl, IX+d, IY+d     |
                                                 |
C-, Z-, S- en V-vlaggen worden beinvloed         |
                                                 |
INC   \  rr              rr = HL, DE, BC, SP,    |
DEC   /                       IX, IY             |
                                                 |
F- of vlag-register wordt NIET beinvloed         |
-------------------------------------------------------------------
```

*figuur 2.7a*

De rekenkundige instrukties zijn onder te verdelen in optel-instrukties (ADD, ADC en INC) en aftrek-instrukties (SUB, SBC en DEC).

De benaming ADD is afkomstig van het engelse werkwoord *add* = *optellen*;

De benaming SUB is afkomstig van het woord *substract* = *aftrekken*;

De benaming INC is afkomstig van het woord *increment* = *verhogen*;

De benaming DEC is afkomstig van het woord *decrement* = *verlagen*.

De 'C' in de instrukties ADC en SBC is afkomstig van het woord *carry*; er wordt mee aangegeven dat de inhoud van het carry-bit in het vlag-register bij de rekenkundige bewerking wordt betrokken.

De betekenis van het begrip 'carry' kan duidelijk worden gemaakt aan de hand van een heel eenvoudig (decimaal) rekensommetje.

Stel we we moeten twee getallen van 1 cijfer bij elkaar optellen en we zijn gedwongen het resultaat ook in de vorm van 1 cijfer weer te geven. Dat zal goed gaan zolang het resultaat van de optelling niet groter is dan 9.
Tellen we echter de getallen 3 en 9 bij elkaar op dan zal ons optel-sommetje er als volgt uitzien:

```
  3
  9 +
----
  2
```

We mogen maar één cijfer gebruiken, hetgeen betekent dat het gebruik 'één onthouden' hier niet op praktische wijze kan worden toegepast. Het resultaat is 2, hetgeen heel duidelijk onjuist is.

Hier komt het gebruik van een vlag (die 0 of 1 kan zijn) ons goed van pas:

we spreken af dat we het vlag-bit hoog (1) maken wanneer er sprake is van een situatie van 'één onthouden' (resultaat groter dan 9); is dat niet het geval dan is de inhoud van dat bit 0 (laag). In ons kleine voorbeeldprogramma zal de vlag hoog (1) zijn, hetgeen betekent dat er 10 opgeteld moet worden bij het eindresultaat.

Omdat een 8-bits-register geen getallen groter dan 255 kan bevatten, zal in het geval van een optelling de carry-vlag hoog (1) worden, wanneer het resultaat groter is dan 255. Dat betekent dat om een juist resultaat te verkrijgen de waarde 256 aan het getal toegevoegd zal moeten worden.

```
Voorbeeld   LD A,200
            ADD A,200
```

Het resultaat (400) is te groot voor het 8-bits-register A. De waarde 256 wordt van het getal afgetrokken en de carry-vlag wordt hoog gemaakt. Register A zal het getal 144 bevatten.

Wanneer we twee getallen van elkaar aftrekken kan een situatie ontstaan, die we kunnen aanduiden met het gezegde 'één lenen'.

```
Voorbeeld   LD A,100
            SUB 200
```

Het getal 200 moet van het getal 100 worden afgetrokken. Getal 2 is te groot - we moeten 'lenen' - en het gevolg is dat de waarde 256 wordt opgeteld bij 100, waarna tot aftrekking wordt overgegaan. Het resultaat dat in de akku verschijnt zal dus zijn: (100 + 256) - 100 = 156. De carry-vlag is hoog (1) om aan te geven dat er werd 'geleend'.

De inhoud van het carry-bit (0 of 1) wordt (zoals hierboven reeds werd aangegeven) bij de berekening betrokken wanneer men de instrukties ADC en SBC gebruikt.
Rekenkundige bewerkingen moeten in dat geval opgesplitst worden in een aantal stappen:

Stel we willen de getallen 100 en 200 optellen en we willen het resultaat in twee 8-bits-registers plaatsen (de enige manier om getallen die groter zijn dan 255 weer te geven!). We moeten dan de vol-

gende stappen zetten:

```
10 ORG 49500
20 LD HL,100
30 LD DE,200     (de getallen worden in register-paren geplaatst)
40 LD A,L        (de lage waarde van HL wordt in A gezet)
50 ADD A,E       (de lage waarde van DE wordt er bij opgeteld)
60 LD C,A        (het resultaat wordt opgeslagen in register C)
70 LD A,H        (de hoge waarde van HL wordt in A gezet)
80 ADC A,D       (de hoge waarde van DE + de inhoud van het carry-bit
                 worden opgeteld bij de inhoud van A)
90 LD B,A        (het resultaat wordt opgeslagen in register B)
```

Registers B en C bevatten nu het eindresultaat. Om de juiste decimale waarde te verkrijgen moet het getal in B (het hoge deel van BC) met 256 vermenigvuldigd worden.

Twee bijzondere rekenkundige instrukties zijn de instrukties INCrement en DECrement. Zij verhogen het betreffende register (of registerpaar) met de waarde 1, INC B betekent bijvoorbeeld dat er bij de inhoud van register B de waarde 1 wordt opgeteld (B = B + 1), of zij verlagen het betreffende register (of registerpaar) met de waarde 1: DEC B betekent dat van de inhoud van register B de waarde 1 wordt afgetrokken (B = B - 1).

Zoals we kunnen zien in figuur 2.7a worden de vlaggen niet beïnvloed wanneer we deze bewerkingen uitvoeren met 16-bits-registers (of registerparen)!

## *2.8 Lijst van gebruikte begrippen*

### Vlaggen

Een vlag is een bit in het vlag- of F-register (zie figuur 1.5c) waarvan de inhoud (0 of 1) afhankelijk is van bepaalde bewerkingen die de Z80-processor uitvoert.

### Carry-vlag

Dit is het laatste bit (bit 0) van het vlag-register. De Z80-instruktie-

set kent twee instrukties die het carry-bit direkt beïnvloeden:

a. SCF (Set Carry Flag):
   maak de carry-vlag hoog (1);

b. CCF (Complement Carry Flag):
   inverteer de inhoud van het carry-bit (d.w.z. 0 wordt 1 en 1 wordt 0).

**Zero-vlag**

Dit is tweede bit (bit 6) van het vlag-register. Wanneer het resultaat van een rekenkundige of logische bewerking gelijk is aan nul dan wordt dit bit hoog (1) gemaakt. Een instruktie waarmee de zero-vlag hoog (1) kan worden gemaakt is de instruktie SUB A (de inhoud van register A wordt afgetrokken van de inhoud van register A, zodat het resultaat altijd nul is; de zero-vlag wordt hoog (1)).

**Sign- of teken-vlag**

Dit is het eerste bit (bit 7) van het vlag-register. Deze vlag wordt gebruikt wanneer men wil rekenen volgens de twee-komplementmethode. In het hoofdstuk dat handelt over logische bewerkingen zullen we daar verder op ingaan.
We kunnen hier reeds stellen dat een getal positief is als de teken-vlag laag (0) is (de getallen 0 - 127). Is de teken-vlag hoog (1) dan is het getal negatief (de getallen 128 - 255).

**P/V-vlag (pariteits- of overflow-vlag)**

Dit is het zesde bit (bit 2) van het vlag-register.

a. Pariteit is 'even' (PE) als het aantal enen in een byte gelijk is aan 2, 4, 6 of 8; de pariteit is 'oneven' (PO) als het aantal enen in een byte gelijk is aan 1, 3, 5 of 7.

b. Wanneer er bij het rekenen volgens de twee-komplementmethode sprake is van overflow wordt bit 2 van het vlag-register hoog (1) gemaakt (PE). (Zie ook figuur 1.5c.)

### 2.9 Logische instrukties

```
------------------------------------------------------------
De instrukties hebben betrekking op de akku (A-register)
------------------------------------------------------------
1  Akku-inhoud wordt veranderd:

OR   \                              r  = A, B, C, D, E, H, L
AND      r, n of (rr)               n  = getal (0 - 255)
XOR  /                              rr = HL, IX+d, IY+d

Z-, C-, S- en Pariteitsvlaggen worden beinvloed
------------------------------------------------------------
CPL of een-komplement              inverteert alle bits
0 wordt 1; 1 wordt 0
Vlag-register wordt niet beinvloed
------------------------------------------------------------
NEG of twee-komplement (CPL+INC)  inverteert alle bits
                                   en voegt 1 toe

Z-, C-, S- en overflow-vlaggen worden beinvloed
------------------------------------------------------------
2  Akku-inhoud wordt niet veranderd:

CP       r, n of (rr)               ComPare = vergelijk

Getal, inhoud register of inhoud geheugenplaats wordt
afgetrokken vn de inhoud van de akku. Is het resultaat 0
(inhoud gelijk) dan wordt de zero-vlag 'geset'.
Is de inhoud A kleiner, dan worden de carry-vlag en de
tekenvlag 'geset' (resultaat negatief).
Ook de overflow-vlag wordt door deze instruktie be-
invloed.
------------------------------------------------------------
```

*figuur 2.9a*

De logische instrukties AND, OR en XOR werden in hoofdstuk 1.8 reeds uitvoerig behandeld. In dat hoofdstuk werd tevens verwezen naar een Basic-programma waarmee deze instrukties kunnen worden getest. In dit gedeelte willen we ons beperken tot een bespreking van de instrukties CPL (komplement), NEG (negatief) en CP

(compare=vergelijk).

## CPL

Figuur 2.9a geeft reeds aan dat een komplement-bewerking alle bits van een byte inverteert. Dat wil zeggen: een bit dat laag (0) is wordt hoog (1); een bit dat hoog (1) is wordt laag (0).
We spreken over een 'één-komplementbewerking'. Daarmee willen we aangeven dat deze bewerking nauw verbonden is met de 'twee-komplementbewerking', die gebruikt wordt bij het binair rekenen met positieve en negatieve getallen. De komplement-bewerking is in feite de eerste stap die tijdens een twee-komplementbewerking gezet moet worden.

## NEG (twee-komplementbewerking)

De NEG (=negatief)-instruktie maakt van een positief getal een negatief getal. De bewerking bestaat uit twee stappen: eerst wordt een komplementbewerking uitgevoerd die alle bits inverteert. Daarna wordt de waarde 1 aan het ontstane getal toegevoegd.

Positieve getallen zijn de getallen 0 tot 127, getallen waarvan het eerste bit (MSB of bit 7) laag is.
Negatieve getallen zijn de getallen 128 tot 255, getallen waarvan het eerste bit (MSB of bit 7) hoog is.

Stel we willen het positieve getal 1 (binair: 00000001) omzetten in het negatieve getal 1. Twee bewerkingen zijn hiervoor nodig:

1. een CPL-bewerking

Alle 0-bits worden 1 en alle 1-bits worden 0. Er ontstaat het volgende binaire resultaat: 11111110. De decimale waarde van dit binaire getal is 254. Willen we de negatieve waarde daarvan achterhalen dan moeten we het getal 256 ervan aftrekken:
254 - 256 = -2. Het resultaat van de één-komplementbewerking is dus onjuist.

2. een INC-bewerking

Het getal wordt verhoogd met de waarde 1. Er ontstaat het volgende

binaire resultaat: 11111111. De decimale waarde van dit binaire getal is 255. De negatieve waarde daarvan is 255 - 256 = -1. Het resultaat van de twee-komplementbewerking is juist!

We zien dus dat het getal 255 volgens de twee-komplement-rekenmethode het negatieve getal -1 voorstelt. Het getal 254 wordt -2; het getal 253 wordt -3; enzovoort...

Het Basic-testprogramma stelt ons in staat de een- en twee-komplementbewerkingen te testen. De getallen worden in hun binaire vorm op het scherm gezet, zodat het bewerkingsproces op een eenvoudige manier te volgen is.

Een probleem waar we mee te maken krijgen bij het rekenen volgens de twee-komplementmethode is het optreden van overflow. Stel we willen twee postieve getallen optellen: de getallen 100 en 50. Het resultaat na een optelling is 150; dat is binair 10010110. We zien dat het teken-bit (bit 7) hoog (1) is; het getal is dus negatief. We spreken in dit geval van een 'overflow-situatie', die wordt aangegeven door de P/V-vlag van het vlag-register (PE).

We kunnen dit kontroleren door (met behulp van de Assembler) het volgende test-programmaatje uit te voeren:

```
10 ' ORG 49500
20 ' LD B,50
30 ' LD A,100
40 ' ADD A,B
50 ' JP PE,BEEP
60 ' RET ;return - naar Basic
70 ' BEEP CALL &H00C0
80 ' RET ;return - naar Basic
90 ' EIND END
```

In regel 50 wordt gekeken of de overflow-vlag hoog is; indien dat het geval is wordt een sprong gemaakt naar de BEEP-routine: Er klinkt een BEEP-toon en het programma stopt.

We weten nu dat de teken-vlag (die aangeeft dat de akku een negatief getal bevat) genegeerd moet worden om het juiste resultaat te verkrijgen. Vlaggen geven ons bij al deze bewerkingen dus sig-

nalen. Een machinetaal-programma kunnen we daarom zien als een spoorbaan, waarop verschillende treintjes rijden, die aan de hand van signaal-lampen een juiste route kunnen rijden. Geven we een verkeerd signaal (groen in plaats van rood) dan zal er onvermijdelijk een katastrofe volgen. Het is daarom erg belangrijk kleine oefenprogrammaatjes te schrijven, waarin het gebruik van de vlaggen wordt getest.

In het hoofdstuk dat handelt over de stapel van de Z80-processor wordt verwezen naar een Basic-testprogramma dat de inhoud van het vlag-register zichtbaar maakt na een rekenkundige bewerking Het staat de nieuwsgierige lezer vrij dat programma alvast uit te proberen.

## CP

In figuur 2.9a zien we dat de compare- of vergelijk-instruktie een bijzondere instruktie is, in die zin dat de inhoud van de akku door deze bewerking niet wordt veranderd. Nadat deze bewerking is uitgevoerd zal echter wel de inhoud van het vlag-register gewijzigd zijn!

Voorbeeld:

```
10 ' ORG 49500
15 ' LD B,100
20 ' LD A,100
25 ' CP B ;B wordt vergeleken met de inhoud van A
30 ' RET M ;return als inhoud A negatief is
35 ' JP C,UIT ;als C-vlag hoog is dan naar UIT
40 ' CALL Z,&H00C3 ;wis scherm als Zero-vlag hoog is
45 ' RET ;return - naar BASIC
50 ' UIT CALL &H00C0 ;BEEP
55 ' RET ;return - naar BASIC
60 ' EIND END
```

Met behulp van deze routine kunnen we drie vlaggen testen: de teken-vlag (P = positief; M = negatief), de carry-vlag en de zero-vlag. De getallen in A en in B (hier gelijk aan elkaar) worden met elkaar vergeleken (regel 25).

Is de teken-vlag hoog (dus de inhoud van A is negatief) dan wordt direkt teruggesprongen naar de Basic-mode (regel 30: RET M). Is de carry-vlag hoog dan wordt een sprong gemaakt naar de routine in regel 50 en er klinkt een beep-toon. Is de carry-vlag laag dan wordt in regel 40 de wis-routine &H00C3 aangeroepen wanneer de zero-vlag hoog is.

In dit voorbeeldprogramma zal inderdaad het laatste gebeuren: de inhouden van A en B zijn namelijk aan elkaar gelijk. Het resultaat van de logische vergelijking is nul; de zero-vlag wordt hoog gemaakt: er ontstaat een Z-situatie.
Zou het resultaat van de vergelijking niet gelijk zijn aan nul dan ontstaat er een NZ-situatie (een situatie waarin de zero-vlag laag (0) is).

Door de waarden in de registers B en A te variëren (regels 15 en 20) kunnen we de verschillende mogelijkheden die het test-programma biedt uitproberen.

## *2.10 Lijst van gebruikte begrippen*

### MSB

Het eerste bit (bit 7 volgens de Z80-bit-notatie) van een byte draagt de naam 'most significant byte' (MSB = meest belangrijke byte). Het laatste bit (bit 0 volgens de Z80-bit-notatie) van een byte draagt de naam 'least significant byte' (LSB = minst belangrijke byte).

### Teken-bit

Het eerste bit (bit 7) van een byte geeft aan of een getal positief (inhoud bit is nul) of negatief (inhoud bit is een) is. Na een rekenkundige of logische bewerking wordt de inhoud van het teken-bit van het byte in register A in het vlag-register geplaatst.
De teken-vlag is hoog (M) als het tekenbit hoog (1) is;
de teken-vlag is laag (P) als het tekenbit laag (0) is (zie figuur 1.5c).

## 2.11 Roteer- en schuif-instrukties

```
---------------------------------------------------------------------
1  Instrukties die alleen betrekking hebben op de akku:

RLA    -   RLCA                       roteer (cyclisch) links
RRA    -   RRCA                       roteer (cyclisch) rechts

Carry-vlag wordt 'geset' (1) als uitgeschoven bit = 1
---------------------------------------------------------------------
2  Algemeen:                          r = A, B, C, D, E, H, L
                                      rr = HL, IX+d, IY+d

RL     -   RLC                        roteer (cyclisch) links
RR     -   RRC   \                    roteer (cyclisch) rechts
                    rr of (rr)
SRL              /                    logische verschuiving:     rechts
SRA                                   rekenkundige verschuiving: rechts
SLA                                   rekenkundige verschuiving: links

Z-, C-, S- en pariteitsvlaggen worden beinvloed
---------------------------------------------------------------------
3  Instrukties ten behoeve van Binary-Code-Decimal rekenen

   (betreffen de registers A en HL)

RRD                                   rotate right digit
RLD                                   rotate left digit
                                      4 bits worden verschoven van (HL)
                                      naar A en omgekeerd

Z-, S- en pariteitsvlaggen worden beinvloed
---------------------------------------------------------------------
```

*figuur 2.11a*

Met behulp van de schuif- en roteer-instrukties kunnen we de bits binnen een byte (naar links of naar rechts) verplaatsen. Laten we als voorbeeld het getal 107 nemen.

De binaire vorm van dit decimale getal is:

```
 128 064 032 016 008 004 002 001
-------------------------------------
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |     64+32+8+2+1 = 107
-------------------------------------
  7   6   5   4   3   2   1   0       bitnummer
```

We schuiven nu alle bits een plaats naar rechts. Er ontstaat het volgende resultaat:

```
-------------------------------------
|   | 0 | 1 | 1 | 0 | 1 | 0 | 1 |
-------------------------------------
```

Het probleem is nu: waar blijft het 'uitgeschoven' bit en wat wordt de inhoud van het eerste bit (bit 7)? De eerste vraag is eenvoudig te beantwoorden: een uitgeschoven bit wordt bij alle schuif- en roteerbewerkingen (met uitzondering van de instrukties RRD en RLD) in het carry-bit van het vlag-register geplaatst (zie figuur 1.5c).

Is het uitgeschoven bit hoog (1) dan zal de carry-vlag hoog zijn (C). Is het uitgeschoven bit laag (0) dan zal de carry-vlag laag zijn (NC).

Met welke waarde de 'opengevallen' plaats wordt opgevuld is afhankelijk van de gebruikte instruktie.

**a. De schuif-instrukties**

Zoals we zien in figuur 2.11a zijn er drie instrukties die een simpele verschuiving (naar links of naar rechts) bewerkstelligen:

SRL (Shift Right Logical) - schuif naar rechts;
SRA (Shift Right Arithmetical) - schuif naar rechts;
SLA (Shift Left Arithmetical) - schuif naar links.

De SRA-instruktie wordt een 'rekenkundige verschuiving' genoemd, omdat bij de verschuiving de inhoud van het eerste bit *niet* verandert. De 'opengevallen' plaats zal dus worden opgevuld met een 1 als de inhoud 'een' was en met een 0 als de inhoud 'nul' was (dit in tegenstelling tot de SRL- en SLA-instrukties, waarbij de inhoud van het vrijgekomen bit wordt opgevuld met de binaire waarde 0). Bekijken we opnieuw het voorbeeld dat hierboven werd

gegeven dan zal na een SRA-bewerking het resultaat de volgende binaire vorm hebben:

```
-------------------------------------------------
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 |    32+16+4+1 = 53
-------------------------------------------------
  7   6   5   4   3   2   1   0      bitnummer
```

Het getal was positief en het blijft positief (het tekenbit, bit 7 is laag). Zou het tekenbit hoog zijn geweest, dan zou het binaire getal 1 in het vrijgekomen bit zijn geplaatst; het negatieve getal zou in dat geval negatief blijven!

***Let op:***
Een bijzonderheid van de schuif- en roteer-instrukties is dat een verschuiving naar rechts een *deling* door twee bewerkstelligt, terwijl een verschuiving naar links een *vermenigvuldiging met twee tot stand brengt.*

*Ook de schuif-instrukties kunnen met een Basic-testprogramma worden getest (zie hoofdstuk 6.5).*

***b. De roteer-instrukties***

Figuur 2.11a laat ons zien dat er twee soorten roteer-instrukties zijn:

1 De 'cyclische' rotatie (RLC, RRC, RLCA en RRCA);
Dit is een 8-bits-rotatie, een uitgeschoven bit wordt dus in het carry-bit van het vlag-register geschoven en tegelijkertijd doorgeschoven naar de opengevallen plaats. De oorspronkelijke inhoud van het byte zal na 8 rotaties weer zijn hersteld.

2 De 'niet-cyclische' rotatie (RL, RR, RLA en RRA);
Dit is een 9-bits-rotatie, een uitgeschoven bit wordt daarbij in het carry-bit van het vlag-register geschoven, terwijl de oude inhoud van het carry-bit wordt doorgeschoven naar de opengevallen plaats. In dit geval zal de oorspronkelijke inhoud van het byte pas na 9 rotaties worden hersteld.

Met het Basic-testprogramma in hoofdstuk 6.6 kunnen de verschillende roteer-instrukties worden uitgetest.

***Let op:***
De roteer-instrukties die alleen betrekking op de akku hebben, werken op dezelfde wijze als de algemene roteer-instrukties. Vanuit een oogpunt van snelheid is het soms beter om de akku-roteer-instrukties te gebruiken, omdat ze twee tot driemaal zo snel werken.

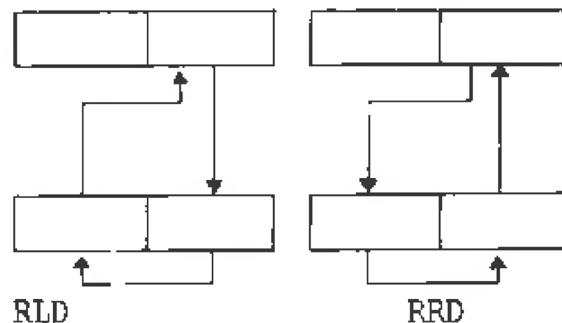
## c. De BCD-roteer-instrukties

Omdat het rekenen volgens de BCD-methode een uiterst gekompliceerde zaak is, zullen we ons hier beperken tot het uiteenzetten van het principe van de 'digit'- of nibble-rotatie.

Bij deze rotaties zijn alleen de rechter-nibble van de akku en de twee nibbles van een lokatie in het RAM-geheugen, waarvan het adres in registerpaar HL staat, betrokken.
Een verschuiving naar rechts zal de linker-nibble doorschuiven naar rechts. De rechter-nibble wordt doorgeschoven naar de akku. De oude inhoud van de akku wordt in de linker-nibble van de geheugen byte geschoven. Een verschuiving naar links doet het omgekeerde (zie figuur 2.11b)

Register A

(HL) geheugenlokatie
aangegeven door HL




*figuur 2.11b*

Voorbeeld:

```
10 ' ORG 49500
20 ' LD A,0 ;inhoud akku is nul
30 ' LD HL,DATA ;HL verwijst naar een geheugenlokatie
40 ' RRD ;roteer nibble naar rechts
50 ' ADD A,48 ;zet inhoud A om in ASCII-waarde
60 ' RST &H18 ;print het getal in A
```

```
70 ' RET ;return - terug naar Basic
80 ' DATA DB 9 ;inhoud rechter-nibble is 9
90 ' EIND END
```

Uitvoering van dit test-programma zal ertoe leiden dat de inhoud van de rechter-nibble van de geneugenlokatie wordt doorgeschoven naar de akku. In regel 50 tellen we de waarde 48 bij bet getal op, zodat de ASCII-waarde van het getal 9 (48 + 9 = 57) wordt gevormd. De printroutine die in regei 60 wordt aangeroepen zet dit getal tenslotte op het scherm.

### *2.12 Lijst van gebruikte begrippen*

**Digit**

Groepje van 4 bits, ook wel nibble genoemd. Zie figuur 1.5a.

**ASCII**

American Standard Code for Information Intercbange; Kode, die aan bepaalde tekens, getallen of symbolen een 8-bits-waarde toekent, die door elke computer (die gebruik maakt van de ASCII-kode) wordt berkend.

### *2.13 Bit-manipulatie- en bit-test-instrukties*

| | |
|---|---|
| BIT b,r of b,(rr)<br><br>test-instruktie,<br>zero-vlag wordt beinvloed<br><br>SET \ b,r of b,(rr)<br>RES /<br><br>bit-manipulatie,<br>vlaggen worden niet beinvloed | b = bitnummer (0 - 7)<br><br>r = A, B, C, D, E, H, L<br>rr = HL, IX+d, IY+d |

*figuur 2.13a*

In hoofdstuk 1.5, 'Bytes en bits', waarin de computer als eenvoudige schakelmachine werd voorgesteld, hebben we deze instrukties reeds ter sprake gebracht, omdat we met behulp van deze instrukties het schakelen op bit-niveau konden imiteren.
De lezer die één en ander alweer vergeten is wordt aangeraden het Basic-proramma waarnaar in hoofdstuk 1.5 wordt verwezen opnieuw uit te testen.

De SET- en RESet-instrukties stellen ons in staat om voorwaarden te scheppen, op grond waarvan een bepaalde routine al dan niet wordt uitgevoerd.
We reserveren bijvoorbeeld de geheugenlokatie 55000 voor het plaatsen van een kontrole- of signaal-byte. Het adres van die lokatie plaatsen we in een van de index-registers (IX of IY). Index-registers (we hebben het reeds eerder opgemerkt) zijn bij uitstek geschikt voor het kreëren van variabelen (vaste lokaties in het geheugen, waarin variabele waarden kunnen worden geplaatst).

Het bron-programma voor de assembler moet als volgt beginnen:

```
10 ' ORG 49500
20 ' START LD IX,55000
30 ' LD (IX+0),0
```

Het index-register IX bevat het adres 55000 (regel 20) en het byte op adres 50000 bevat de waarde 'nul' (regel 30), d.w.z. dat alle bits in de 'RESet'- of uit-stand staan.

We ontwerpen vervolgens een test-procedure, die moet voeren naar twee keuze-mogelijkheden: de byte op adres 55000 *wel* of *niet* beïnvloeden. Als voorbeeld kiezen we een procedure, die ons in staat stelt te bepalen of de computer een Basic-programma bevat.

We weten (zie hoofdstuk 1.1) dat het Basic-programma in het geheugen wordt opgeslagen vanaf adres 32769. Wanneer er geen bronprogramma aanwezig is, dan zullen de adressen 32769 en 32770 de waarde 'nul' bevatten; in het andere geval bevatten deze adressen het startadres van de volgende programmaregel.

Adres 32769 bevat (wanneer een programma aanwezig is) de lage byte van dat startadres; adres 32770 bevat de hoge byte. Omdat

Basic-programma's altijd geplaatst worden in een RAM-gebied waarvan het startadres hoger is dan 32768, zal de inhoud van de byte op adres 39770 altijd groter zijn dan 127 (32768 gedeeld door 256).
Vinden we nu de waarde 'nul' op dit adres dan kunnen we de konklusie trekken, dat er momenteel geen Basic-programma aanwezig is.

We voegen daarom aan het assembler-bronprogramma de volgende regels toe:

```
40 ' LD A,(32770) ;inhoud lokatie 32770 in A
50 ' CP 0 ;kijk of inhoud nul is
60 ' JP Z,EIND ;ja, dan naar EIND
70 ' SET 0,(IX+0)
80 ' EIND RET
90 ' END
```

In regel 50 bekijken we of geheugenlokatie 32770 de waarde 'nul' bevat. Wanneer dat het geval is dan wordt een sprong gemaakt naar de instruktie in regel 80 "return", dat wil zeggen: keer terug naar de direkte Basic-mode. Is de inhoud niet gelijk aan 'nul' dan wordt de instruktie in regel 70 uitgevoerd. bit 0 van de byte in geheugenlokatie 55000 wordt dus hoog (1) gemaakt.

We hebben nu een voorwaarde geschapen, op grond waarvan we de computer bepaalde handelingen wel of niet kunnen laten verrichten.

Als voorbeeld toetsen we in de direkte Basic-mode de volgende opdrachten in:

```
DEFUSR0=49500:Z=USR0(0)
IF PEEK(55000)=0 THEN PRINT"VRIJ" ELSE PRINT"PROGRAMMA"
```

Wanneer een Basic-programma (of assembler-bron-programma) aanwezig is dan zal het 'voorwaardelijke' Basic-kommando 'IF...THEN' de mededeling "PROGRAMMA" op het scherm plaatsen; wordt dat programma door het kommando NEW verwijderd en de opdracht opnieuw uitgevoerd, dan zal de mededeling "VRIJ" worden afgedrukt.

Konklusie:
het byte op adres 55000 kan ons inlichtingen verschaffen over een bepaalde situatie. De stand van bit 0 komt overeen met de woorden 'ja' en 'nee'. Is het bit laag (0) dan is het antwoord 'nee'; is het bit hoog (1) dan is het antwoord 'ja'. Bij het programmeren in machinetaal maken we voortdurend gebruik van deze elementaire 'ja/nee-situaties'!

## *2.14 Lijst van gebruikte begrippen*

**Voorwaarde**

Wanneer een machinetaal-programma is opgebouwd uit meerdere routines dan zullen er bepaalde signalen aanwezig moeten zijn, op grond waarvan de computer bepaalde keuzes kan maken: we kunnen ook zeggen er moeten voorwaarden worden geschapen, op grond waarvan keuzes worden gemaakt. Die woorwaarden worden geschapen door de Z80-processor, door het setten en resetten van de hits in het vlag-register (zie figuur 1.5c), maar we kunnen ook zelf die voorwaarden in het leven roepen, door vaste lokaties in het geheugen als 'signaal-byte' te gebruiken.

**Voorwaardelijke instrukties**

Instrukties die bepaalde handelingen alleen uitvoeren wanneer aan zekere voorwaarden is voldaan. Bijvoorbeeld het Basic-kommando 'IF..THEN' of de machinetaal-instruktie 'CALL Z,..' (als de zero-vlag hoog is roep dan een subroutine aan).

```
--------------------------------------------------------------
1  Direkte adressering           2  Relatieve adressering

   a  onvoorwaardelijk
   CALL  -  JP          \              JR       \
                          nn                      d
   b  voorwaardelijk    /                        /
   CALL IF,  -  JP IF,                 JR IF,

   IF=Z, NZ, C, NC, PE, PO, P of M     IF=Z, NZ, C,NC
-------------------------------+------------------------------
                               | DJNZ   d
                               | Dec B - Jr NZ,d (B=teller)
-------------------------------+------------------------------
nn = getal/adres (0 - 65535)   | d = verplaatsing
                               | positief: 0 - 127
                               | negatief: 1 - 128
-------------------------------+------------------------------
PE=even pariteit/              |
   overflow          (P/V=1)   | M = minus/negatief    (S=1)
PO=oneven pariteit (P/V=0)     | P = positief          (S=0)
Z= zero              (Z=1)     | C = carry             (C=1)
NZ=not zero          (Z=0)     | NC = not carry        (C=0)
-------------------------------+------------------------------
```

*figuur 2.15a*

De sprong- en subroutine-instrukties kunnen worden onderverdeeld in onvoorwaardelijke instrukties (jump, jump relative, call en restart) en voorwaardelijke instrukties (b.v. JP Z,adres - spring naar een adres als de zero-vlag hoog is).

De voorwaarden waaraan moet worden voldaan liggen besloten in het vlag-register van de Z80-processor (figuur 1.5c). Hierin worden tijdens rekenkundige, logische en 'schuif'-bewerkingen bepaalde bits (afhankelijk van de bewerking) hoog of laag gemaakt. Deze bits worden tijdens de uitvoering van de voorwaardelijke instrukties getest.

In dit boek beperken we ons tot die signaal-bits, die met behulp van

de voorwaardelijke Z80-instrukties kunnen worden uitgetest: het pariteits (of overflow)-bit, het zero-bit, het teken- (of sign-)bit en het carry-bit.

a Er is sprake van een PE-situatie wanneer het pariteits-bit van het vlag-register hoog is (er is in dat geval ofwel sprake van een overflow-toestand (zie hoofdstuk 2.9b) of van een toestand waarin het aantal bits in een byte 'even' is. Er is sprake van een PO-situatie wanneer het pariteits-bit laag is (aantal bits oneven/geen overflow).

b Er is sprake van een Z of zero-situatie wanneer het zero-bit van het vlag-register hoog is (het resultaat van een uitgevoerde bewerking is 'nul').
Er is sprake van een NZ of not-zero-situatie wanneer het zero-bit laag is (het resultaat van een bewerking is 'niet-nul').

c Er is sprake van een M of minus-situatie wanneer het teken-bit van het vlag-register hoog is (het getal dat zich in de akku bevindt is volgens de twee-komplementmethode negatief).
Er is sprake van een P of positief-situatie wanneer het teken-bit laag is (het getal is positief).

d Er is sprake van een C of carry-situatie wanneer het carry-bit van het vlag-register hoog is (zie figuur 1.5c en hoofdstuk 2.7).
Er is sprake van een NC of not-carry-situatie wanneer het carry-bit laag is.

Zoals we kunnen zien in figuur 2.15a kunnen met behulp van de voorwaardelijke call- en jump-instrukties alle mogelijkheden worden uitgetest.
In figuur 2.15a zien we ook dat de voorwaardelijke instrukties CALL en JP worden omschreven als 'CALL IF,' en 'JP IF,'. Daarmee willen we wijzen op de overeenkomsten met de Basic-sprong-en subroutine-instrukties.

De voorwaardelijke JP-instruktie kan worden vergeleken met de Basic-sprong-opdracht 'IF...THEN GOTO'.

Een uitzondering moeten we daarbij maken voor de 'relatieve sprong-instruktie': JR.

Bij een relatieve sprong wordt er niet rechtstreeks naar een adres gesprongen, maar wordt een 'stapsgewijze sprong' gemaakt, waarbij het aantal stappen (als verplaatsingswaarde) wordt opgenomen in de byte die volgt op de instruktie.

Wie programmeert met behulp van een assembler-vertaal-programma hoeft deze verplaatsingswaarde niet zelf uit te rekenen. Hij hoeft alleen maar te verwijzen naar een label, waarna de assembler het rekenwerk verricht.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD A,42 ;*
30 ' JR PRINT
40 ' RET ;return
50 ' PRINT RST &H18
60 ' RET ;return
70 ' EIND END
```

In regel 50 wordt aan de PRINT-opdracht 'RST &H18' het label 'PRINT' toegevoegd, zodat de assembler in regel 30 weet waarheen gesprongen moet worden. De assembler berekent de afstand (het aantal bytes tussen de twee instrukties) en plaatst de gevonden waarde in het geheugen. (Met behulp van de dis-assembler kan men de berekende waarden, wanneer men dat wil, nitlezen.)

De voorwaardelijke call-instruktie komt overeen met de Basic-opdracht 'IF..THEN GOSUB'. Een call-instruktie verwijst dus naar een subroutine: een routine die geen deel uitmaakt van het hoofdprogramma en die daarom altijd moet worden afgesloten met een return-kommando, om daarmee de voortzetting van het hoofdprogramma mogelijk te maken.

Een bijzondere subroutine-instruktie is de restart-instruktie. Het kenmerkende van zo'n RST-instruktie is dat hij gebonden is aan een aantal vaste start-adressen, die verwijzen naar standaard-routines in het ROM-geheugen:

1. **RST &H00**:

CHKRAM - uitvoeren van een warme start (RESET);

**2. RST &H08:**

SYNCHR - vergelijkt een karakter in een Basic-programma (aangegeven door HL) met een kontrole-karakter dat volgt op de RST-instruktie;

**3. RST &H10:**

CHRGTR - leest een karakter (waarvan het adres in HL staat) in het programmageheugen en plaatst dat in de akku;

**4. RST &H18:**

OUTDO - print-routine;

**5. RST &H20:**

DCOMPR - vergelijkt de inhouden van de registers DE en HL en legt het resultaat vast in het vlag-register:

Z = 1 als HL en DE gelijk zijn aan elkaar,
C = 1 als HL kleiner is dan DE,
C = 0 (NC) als HL groter is dan DE;

**6. RST &H28:**

GETYPR - onderzoekt welk variabelentype er door de Basic-interpreter wordt verwerkt en plaatst het resultaat op de volgende wijze in register A:

A = 255: Integere variabele,
A = 0: String-variabele,
A = 1: Variabele met enkele precisie,
A = 5: Variabele met dubbele precisie;

**7. RST &H30:**

CALLF - roept een routine aan in een bepaald Slot. in dit boek wordt de Slot-opbouw van de MSX-computer niet behandeld;

**8. RST &H38:**

KEYINT - Interrupt-routine, die het toetsenbord aftast om te kijken welke toets (of groep van toetsen) er is ingedrukt.

## *2.16 Lijst van gebruikte begrippen*

### Slot

De Z80-microprocessor kan een geheugengebied van 65535 bytes besturen. Dat geheugen bevindt zich in geheugenbanken of sloten. Een geheugenbank of slot kan ROM-geheugen bevatten, RAM-geheugen, of kan gewoon beschikbaar zijn voor uitbreidingen, zoals disk-drive en spelmodules.
Via 'slot-selektie' worden gedeelten van de geheugenbanken binnen het bereik van de processor geschakeld. Omdat de slot-opbouw van de MSX-computer in dit boek niet wordt behandeld verwijzen we hier naar een publikatie waarin dit onderwerp vrij uitvoerig aan de orde komt:
*MSX computer magazine, 2e jaargang nummer 12, 'MSX geheugenstrukturen'.*

### Standaard-routines

Routines die deel uitmaken van het BIOS-ROM van de MSX-computer. Deze routines hebben vaste toegangsadressen, die voor alle computers die onder de MSX-norm vallen gelijk zijn.

Machinetaal-programma's die gebruik maken van deze routines zullen daarom op alle MSX-machines kunnen werken. Zie voor een overzicht van alle standaard-BIOS-ROM-routines:

1. MSX2 Zakboekje, Wessel Akkermans, uitgeverij Stark-Texel b.v. 1987;

2. MSX-ROM-BIOS-handboek (vertaling van: The MSX Red Book), uitg: Terminal Software Publicaties, 1986.

Het betreft hier routines die zowel op de MSX als de MSX2 computers werken.

## 2.17 Stack-(stapel)instrukties

```
-----------------------------------------------------------
PUSH       \  rr                rr = AF, HL, BC, DE, IX, IY
POP        /

LD SP,     \  rr                rr = HL, IX, IY
EX (SP),   /

F- of vlag-register wordt niet beinvloed
-----------------------------------------------------------
```

*figuur 2.17a*

De MSX-computer reserveert een bepaald gebied in het RAM-geheugen voor de tijdelijke opslag van de inhouden van Z80-registerparen. Dit gebied wordt de stack (stapel) genoemd (zie hoofdstuk 2.6). De 8-bits-registers kunnen ***niet*** afzonderlijk in dit gebied worden geplaatst. Dat betekent dat het bewaren van de inhoud van register A alleen kan gebeuren in kombinatie met de inhoud van het F- of vlag-register.

Het plaatsen van register-inhouden in het geheugen gebeurt met de Z80-instruktie PUSH rr. Een push-instruktie bekijkt de inhoud van het stapel-register (de stack-pointer) en plaatst de gegevens in het geheugen vanaf het adres dat door dit register wordt aangewezen. Daarbij wordt eerst de inhoud van de stapel-wijzer (of stack-pointer) met 1 verlaagd.
Vervolgens wordt het hoge deel van het registerpaar in het geheugen gezet. Daarna wordt de stapel-wijzer opnieuw met 1 verlaagd en het lage deel wordt in het geheugen gezet.

Het uitlezen van het stapel-geheugen gebeurt met behulp van de Z80-instruktie POP rr. Hierbij gebeurt het omgekeerde:
De inhoud van de geheugenlokatie waarnaar de stapel-wijzer verwijst wordt in het lage deel van het registerpaar geladen. De stapel-wijzer wordt met 1 verhoogd en de inhoud van de volgende geheugenlokatie wordt in het hoge deel van het register-paar geladen, waarna de stapel-wijzer opnieuw met 1 wordt verhoogd.

De stack-pointer is een 16-bits-register, dat vanuit het machinetaal-programma beïnvloed kan worden. Zo kan men met behulp van de

instruktie 'LD SP,adres' de stapel op een andere plaats in het geheugen positioneren. In een testprogramma dat achter in dit boek is opgenomen (hoofdstuk 6.7) wordt gebruik gemaakt van deze instruktie, om het mogelijk te maken het vlag-register van de Z80-processor (zie figuur 1.5c) uit te lezen.

Zoals we reeds hebben opgemerkt wordt het A-register tesamen met het F-(vlag)-register in het stapel-geheugen gezet. Dat betekent dat we na een bepaalde bewerking de inhoud van beide registers in een door de stapel-wijzer aangegeven geheugenlokatie kunnen zetten, zodat we de inhouden met behulp van de Basic-uitlees-funktie PEEK op het beeldscherm kunnen plaatsen. Op die manier kan de werking van allerlei rekenkundige en logische bewerkingen worden bestudeerd; hetgeen voor een juist begrip van machinetaal-programmering onontbeerlijk is.

Het start-adres van de stapel wordt ook wel de 'top' van de stapel genoemd, om daarmee aan te geven dat de Stapel als het ware omlaag klimt in het geheugen wanneer er informatie op wordt ge*PUSH*ed.

Voorbeeld:

```
10 LD SP,50000 ;startadres stapel wordt 50000
20 LD BC,&H1250 ;hoge deel: &H12 - lage deel:&H50
30 PUSH BC ;plaats BC in het stapel-geheugen
```

Het resultaat zal het volgende zijn:

```
               50000  startadres (TOP) stapel
PUSH  verlaag: 49999  inhoud B = &H12 in lokatie 49999
      verlaag: 49998  inhoud C = &H50 in lokatie 49998
```

De inhoud van de stapel-wijzer zal na voltooiing van de push-instruktie 49998 zijn (SP = 49998).
De POP-instruktie doet het omgekeerde:

```
               49998  inhoud stapel-wijzer
POP            49998  inhoud lokatie 49998 in L
     verhoog:  49999  inhoud lokatie 49999 in H
     verhoog:  50000
```

De inhoud van de stapel-wijzer zal na voltooiing van de pop-instruktie 50000 zijn (SP = 50000); de oorspronkelijke waarde is hersteld!

We zien dus dat we de informatie van registerpaar BC in het stapel-geheugen kunnen deponeren, waarna we met een 'pop HL'-instruktie de inhoud van de betreffende lokaties (dat is de inhoud van BC) in registerpaar HL kunnen laden.
Deze 'kunstgreep' maakt het ons mogelijk om binnen een machinetaal-programma registers aan elkaar gelijk te maken!

Het werken met stapel-instrukties is op zichzelf tamelijk eenvoudig; moeilijkheden doen zich voor wanneer men op onoordeelkundige wijze push- en pop-instrukties door elkaar gaat gebruiken.

Wanneer we werken met de stapel dan moeten we ons realiseren dat we in die gevallen samen moeten werken met de Z80-processor; immers: de Z80-processor gebruikt de stapel om er 'return- of terugkeer-adressen' in op te slaan.
Iedere keer wanneer er een subroutine wordt aangeroepen in een machinetaal-programma wordt een terugkeer-adres op de stapel gezet, om een voortzetting van het hoofd-programma na een return-instruktie mogelijk te maken.

Gebruiken we binnen een subroutine een push-instruktie dan zal de stapel-wijzer niet meer verwijzen naar dit terugkeer-adres! Een push-instruktie zal binnen een subroutine dan ook *altijd* gevolgd moeten worden door een pop-instruktie. Alleen op die manier is de computer in staat terug te springen naar de juiste plaats in het programma!

Een simpel rekenprogrammaatje stelt ons in staat het juiste gebruik van push- en pop-instrukties te kontroleren: Het aantal push-instrukties binnen een subroutine moet gelijk zijn aan het aantal pop-instrukties.

Voorbeeld:

```
10 PUSH HL
20 PUSH DE
30 PUSH BC
40 CALL &H00C0 ;BEEP
50 POP BC
60 POP DE
70 POP HL
80 RET
```

De routine begint met drie push-instrukties en eindigt met hetzelfde aantal pop-instrukties. De inhoud van de stapel-wijzer wordt door die konstruktie niet veranderd.

Omdat binnen onze voorbeeld-routine de inhoud van BC het laatst in het stapel-geheugen werd gezet zullen die geheugenlokaties het eerst moeten worden uitgelezen met behulp van de 'POP BC'-instruktie. De engelse benaming hiervoor is 'LIFO' (last in - first out, d.w.z wat er het laatst in werd geplaatst zal er het eerst uit moeten worden gehaald).

### *2.18 Blok-zoek- en verplaats-instrukties*

```
-------------------------------------------------------
1  Verplaats-instrukties:

LDI   -  LLD                      Load + inc/dec
LDIR  -  LDDR                     Load + inc/dec + repeat

P/V-(pariteits/overflow)vlag wordt beinvloed:
P/V=1 (schrijf PE) als BC<>0 - P/V wordt 0 (PO) als BC=0
-------------------------------------------------------
2  Zoek-instrukties:

CPI   -  CPD                   Compare + inc/dec
CPIR  -  CPDR                  Compare + inc/dec + repeat

Zero-vlag wordt geset (1) als A=(HL);
P/V vlag wordt 0 (schrijf PO) als BC=0
-------------------------------------------------------
```

*figuur 2.18a*

## a. Verplaats-instrukties

Met behulp van de blok-verplaats-instrukties kunnen blokken gegevens binnen het RAM-geheugen (of van het ROM-geheugen naar het RAM-geheugen) worden verplaatst.

1. LDIR (Load - inc - repeat)
   Het begin-adres van het blok dat verplaatst moet worden wordt in register-paar HL geplaatst. Het doel-adres wordt in register-paar DE verwacht. De lengte van het blok moet in register-paar BC worden geladen.
   Deze instruktie voert in feite de volgende bewerkingen uit:

```
LUS LD A,(HL)   inhoud lokatie aangegeven door HL in
                de akku
LD (DE),A       inhoud akku in lokatie aangegeven door
INC HL          HL=HL+1                             DE
INC DE          DE=DE+1
DEC BC          BC=BC-1
JP PE,LUS       herhaal tot BC = 0
```

2. LDI (Load - inc)
   Deze instruktie werkt volgens hetzelfde principe. Alleen ontbreekt nu de herhaal-opdracht. Men zal dus zelf een programmalus moeten ontwerpen, waarin deze instruktie is opgenomen. Men kan daarbij gebruik maken van de pariteits/overflow-vlag: die zal hoog zijn zolang BC < > 0 (zie figuur 2.18a).

3. LDDR (Load - dec - repeat)
   Het eind-adres van het blok dat verplaatst moet worden wordt in registerpaar HL gezet. DE bevat het doel-adres (dat tevens het laatste adres van het verplaatste blok zal zijn). BC bevat de lengte van het blok.
   Het verplaatsingsprincipe is weer hetzelfde; alleen wordt in dit geval de inhoud van alle registerparen met 1 verlaagd (net zolang tot de inhoud van BC gelijk is aan 'nul').

4. LDD (Load - dec)
   Zie het kommentaar bij LDI.

**b. Zoek-instrukties**

Met behulp van de blok-zoek-instrukties kan een bepaalde waarde in een geheugenblok worden opgezocht. Wanneer de betreffende waarde wordt gevonden zal dat kenbaar worden gemaakt door het hoog-maken van de zero-vlag.
We kunnen deze instruktie vergelijken met de Basic-funktie INSTR, die het ons mogelijk maakt een bepaald karakter op te zoeken in een string. Het principe is ongeveer hetzelfde, maar de mogelijkheden zijn beperkter (een vanzelfsprekende zaak in feite, omdat Basic een hogere programmeertaal is: dus gebruik maakt van kant en klare machinetaal-routines, die wij als programmeur zelf moeten schrijven).

1. CPIR (Compare - inc - repeat)
   Het begin-adres van het blok dat onderzocht dient te worden moet in registerpaar HL worden geplaatst.
   De lengte van het blok wordt in registerpaar BC verwacht. De akku moet de op te zoeken waarde bevatten. Deze instruktie voert in feite de volgende handelingen uit:

```
LUS CP (HL)              vergelijk inhoud A met inhoud
                         geheugen.lokatie
JP Z,CONTIF A = (HL)     THEN EIND
INC HL                   HL=HL+1
DEC HL                   BC=BC-1
JP PE,LUS                herhaal zolang BC niet gelijk is
                         aan nul
CONT                     zet het programma voort
```

   Wordt de gezochte waarde gevonden, dan stopt de zoek-routine en de zero-vlag wordt hoog gemaakt. Het adres van de gezochte waarde is HL-1.

2. CPI (Compare - inc)
   Zie kommentaar bij LDI.

3. CPDR (Compare - dec - repeat)
   Werkt net als de CPIR-instruktie, alleen wordt nu de inhoud van registerpaar HL verlaagd.
   Wordt de gezochte waarde gevonden dan stopt de routine en de

zero-vlag wordt hooggemaakt. Het adres van de gezochte waarde is in dat geval HL+1.

4. CPD (Compare - dec)
   Zie kommentaar bij LDI.

Voorbeeld bij a.1 (LDIR):

```
10 ' ORG 49500
20 ' LD HL,32776 ;startadres tekst-regel 10
30 ' LD DE,50000 ;doeladres
40 ' LD BC,10 ;lengte blok
50 ' LDIR ;verplaats de tekstregel
60 ' RET ;return - naar Basic
70 ' EIND END
```

Na uitvoering van dit programma zal de tekst " ORG 49500" (in de vorm van ASCII-tekens) zich bevinden in het geheugengebied met het startadres 50000, dat we uit kunnen lezen met behulp van de Basic-PEEK-funktie:

```
FOR I = 50000 TO 50010:PRINT CHR$(PEEK(I));:NEXT
```

Voorbeeld bij b.1 (CPIR):

```
10 ' ORG 49500
20 ' LD HL,32776
30 ' LD BC,10 ;lengte blok
40 ' LD A,57 ;ASCII-kode voor het getal 9
50 ' CPIR ;zoek
60 ' LD (MEM),HL ;plaats inhoud HL in geheugen
70 ' RET ;return - naar BASIC
80 ' MEM DS 2
90 ' EIND END
```

Na uitvoering van dit programma zal de waarde van HL in het geheugen staan: en wel in de lokaties die worden aangegeven door het label MEM (zie daarvoor de assembler-listing). Met behulp van de Basic-PEEK-funktie lezen we die lokaties uit:

```
PRINT PEEK(MEM) + 256*PEEK(MEM+1),
```

**b. Zoek-instrukties**

Met behulp van de blok-zoek-instrukties kan een bepaalde waarde in een geheugenblok worden opgezocht. Wanneer de betreffende waarde wordt gevonden zal dat kenbaar worden gemaakt door het hoog-maken van de zero-vlag.
We kunnen deze instruktie vergelijken met de Basic-funktie INSTR, die het ons mogelijk maakt een bepaald karakter op te zoeken in een string. Het principe is ongeveer hetzelfde. maar de mogelijkheden zijn beperkter (een vanzelfsprekende zaak in feite, omdat Basic een hogere programmeertaal is: dus gebruik maakt van kant en klare machinetaal-routines, die wij als programmeur zelf moeten schrijven).

1. CPIR (Compare - inc - repeat)
   Het begin-adres van het blok dat onderzocht dient te worden moet in registerpaar HL worden geplaatst.
   De lengte van het blok wordt in registerpaar BC verwacht. De akku moet de op te zoeken waarde bevatten. Deze instruktie voert in feite de volgende handelingen uit:

```
LUS CP (HL)                 vergelijk inhoud A met inhoud
                            geheugen.lokatie
JP Z,CONTIF A = (HL)        THEN EIND
INC HL                      HL=HL+1
DEC HL                      BC=BC-1
JP PE,LUS                   herhaal zolang BC niet gelijk is
                            aan nul
CONT                        zet het programma voort
```

   Wordt de gezochte waarde gevonden, dan stopt de zoek-routine en de zero-vlag wordt hoog gemaakt. Het adres van de gezochte waarde is HL-1.

2. CPI (Compare - inc)
   Zie kommentaar bij LDI.

3. CPDR (Compare - dec - repeat)
   Werkt net als de CPIR-instruktie, alleen wordt nu de inhoud van registerpaar HL verlaagd.
   Wordt de gezochte waarde gevonden dan stopt de routine en de

zero-vlag wordt hooggemaakt. Het adres van de gezochte waarde is in dat geval HL+1.

4. CPD (Compare - dec)
   Zie kommentaar bij LDI.

Voorbeeld bij a.1 (LDIR):

```
10 ' ORG 49500
20 ' LD HL,32776 ;startadres tekst-regel 10
30 ' LD DE,50000 ;doeladres
40 ' LD BC,10 ;lengte blok
50 ' LDIR ;verplaats de tekstregel
60 ' RET ;return - naar Basic
70 ' EIND END
```

Na uitvoering van dit programma zal de tekst " ORG 49500" (in de vorm van ASCII-tekens) zich bevinden in het geheugengebied met het startadres 50000, dat we uit kunnen lezen met behulp van de Basic-PEEK-funktie:

```
FOR I = 50000 TO 50010:PRINT CHR$(PEEK(I));:NEXT
```

Voorbeeld bij b.1 (CPIR):

```
10 ' ORG 49500
20 ' LD HL,32776
30 ' LD BC,10 ;lengte blok
40 ' LD A,57 ;ASCII-kode voor het getal 9
50 ' CPIR ;zoek
60 ' LD (MEM),HL ;plaats inhoud HL in geheugen
70 ' RET ;return - naar BASIC
80 ' MEM DS 2
90 ' EIND END
```

Na uitvoering van dit programma zal de waarde van HL in bet geheugen staan: en wel in de lokaties die worden aangegeven door het label MEM (zie daarvoor de assembler-listing). Met behulp van de Basic-PEEK-funktie lezen we die lokaties uit:

```
PRINT PEEK(MEM) + 256*PEEK(MEM+1),
```

waarna we kunnen kontroleren of het getal 9 (ASCII-waarde: 57) inderdaad te vinden is op het aangegeven adres (verminderd met de waarde 1).

## *2.19 Input (invoer)- en output (uitvoer)-instrukties*

```
------------------------------------------------------------
1  Instrukties die betrekking hebben op de akku:

IN (A),n                      n = getal (0 - 255)
OUT (n),A                     register A bevat poort-nummer

F- of vlag-register wordt niet beinvloed
------------------------------------------------------------
2  Instrukties die betrekking hebben op het C-register:

OUT (C),r                     r = A, B, C, D, E, H, L
Vlaggen worden niet
beinvloed                     register C bevat poort-nummer

                              OUT verwijst naar uitvoer
Z-, S- en pariteits-          IN verwijst naar invoer
vlaggen worden beinvloed
------------------------------------------------------------
3  Blok-input/ouput-instrukties:

OUTI    -    OUTD             OUT + inc/dec
OTIR    -    OTDR             OUT + inc/dec + repeat

INI     -    IND              IN + inc/dec
INIR    -    INDR             IN + inc/dec + repeat

Register B bevat het aantal invoer/uitvoer-opdrachten
 (maximaal 255)
Register C bevat het poort-nummer
Registerpaar HL bevat het geheugenlokatie-adres

Z-vlag wordt beinvloed        is B<>0 dan NZ (Z=0)
                              is B=0  dan  Z (Z=1)
------------------------------------------------------------
```

*figuur 2.19a*

De Z80-microprocessor kan rechtstreeks een geheugengebied van 65535 bytes 'besturen'. Moet informatie in- of uitgevoerd worden naar gebieden die buiten dat 'vaste' geheugenterrein vallen, dan zullen die gebieden ofwel binnen het bereik van de processor geschakeld moeten worden (we spreken in dat geval over slot-selektie of 'bank-switching'), ofwel er zullen speciale verbindingskanalen in werking moeten worden gesteld, waarlangs informatie-vervoer kan plaatsvinden. Dergelijke verbindingskanalen noemen we poorten.
Zo kan de Z80-processor via een speciale video-poort kommuniceren met het video-RAM: De VDP-data-poort. Het nummer van die data-poort kunnen we achterhalen door adres 6 van het het geheugen uit te lezen.
Dat gaat aldus: PRINT HEX$(PEEK(6)).
De meeste MSX-computers zullen de waarde 98 op het scherm zetten.

Alleen via deze data-poort kan het video-RAM van de gemiddelde MSX-computer benaderd worden. De in- en uitvoer via poorten kan gestuurd worden met behulp van de IN- en OUT-instrukties van de Z80-processor.
Zelfs in Basic kunnen de poorten direkt benaderd worden met behulp van de Basic-instrukties 'INP' en 'OUT'.

Omdat de poort-nummering niet voor alle MSX-computers gelijk hoeft te zijn, (in de praktijk zullen de verschillen miniem zijn) wordt een direkte besturing van rand-apparatuur (waaronder ook het video-scherm valt) door verschillende auteurs afgeraden. Degene echter die in de eerste plaats wil experimenteren met zijn eigen computer kan deze instrukties gerust in zijn programma opnemen.

Wil men programma's schrijven die op alle computers moeten kunnen werken dan kan men beter gebruik maken van de standaard-routines waarover het MSX-BIOS-ROM beschikt.
De belangrijkste routines worden in het volgende hoofdstuk, dat Basic-instrukties naast overeeukomstige machinetaal-routines plaatst, uitvoerig besproken.

Voor diegenen, die, gedreven door nieuwsgierigheid, toch de poort-instrukties uit willen proberen is er een testprogramma opgenomen, waarmee de Z80-OUT-instruktie en de Basic-OUT-instruktie kunnen worden uitgeprobeerd (hoofdstuk 6.8).

IN- en OUT-instrukties moeten, wanneer in- of uitvoer van meerdere gegevens is vereist, in een programma-lus worden opgenomen. (In het volgende hoofdstuk worden programma-lussen uitvoerig besproken in het gedeelte waarin het Basic-kommando FOR...NEXT wordt behandeld).

De Z80-instruktie-set kent echter ook instrukties die een blokverplaatsing mogelijk maken. In figuur 2.19a zien we deze instrukties in deel III. Het aantal 'invoeren' of 'uitvoeren' (dus de lengte van het blok) wordt in het geval van blok-verplaatsing in register B geplaatst. Registerpaar HL moet het begin-adres van het geheugenblok dat in het VRAM geschreven moet worden (of waarnaartoe gegevens geschreven moeten worden) bevatten.

Wanneer we gegevens naar het Video-RAM willen schrijven zullen we twee poorten moeten instrueren:

1. de DATA-poort, voor de meeste MSX-computers poort &H98;
2. de ADRES-poort: de daaropvolgende poort &H99.

Voordat we informatie op het scherm plaatsen moet het startadres (dus het adres dat aangeeft waar op het scherm de informatie moet verschijnen) worden doorgegeven aan de video-processor. Dat adres laden we in registerpaar HL en via twee OUT-opdrachten geven we het door (eerst het lage byte - daarna het hoge byte) aan de videoprocessor. Daarna kan het blok gegevens via de DATA-poort naar het video-RAM worden getransporteerd.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD C,&H99 ;ADRES-poort-nummer in register C
30 ' LD HL,680 ;startregel bij 40-tekens-scherm is 17
40 ' OUT (C),L ;lage deel HL naar Video-processor
50 ' OUT (C),L ;hoge deel HL naar Video-processor
60 ' LD C,&H98 ;DATA-poort-nummer in register C
70 ' LD B,120 ;lengte van het blok
80 ' LD HL,32776 ;begin-adres RAM-geheugenblok
90 ' OTIR ;schrijf 120 bytes naar het Video-RAM
95 ' RET ;return - naar BASIC
99 ' EIND END
```

Wanneer we dit programma uitvoeren zal de tekst van de eerste twee Basic-programma-regels op het scherm verschijnen (er zullen ook allerlei vreemde tekens worden afgebeeld, op die plaatsen waar de programmaregels regelnummers en Basic-tokens bevatten). Door de waarde van HL in regel 30 te variëren kan de plaats op het scherm gewijzigd worden.

Wil men het programma helemaal 'professioneel' maken dan zullen de interrupt-instrukties 'EI' (enable interrupt) en 'DI' (disable interrupt) toegevoegd moeten worden:

```
35 ' DI
55 ' EI
```

Het principe van de IN- en OUT-instrukties is dus in wezen niet zo moeilijk. In de praktijk echter zal men op allerlei problemen stuiten. In die gevallen is het beter zijn toevlucht te nemen tot de standaard-routines in het BIOS-ROM, die per slot van rekening zijn ontworpen om het programmeren te vergemakkelijken.

### *2.20 Enige rest-instrukties*

In dit hoofdstuk hebben we de Z80-instrukties ondergebracht in een aantal hoofd-kategorieën, om op die manier wat meer greep te kunnen krijgen op het geheel.

Wie gekonfronteerd wordt met een boek waarin honderden instrukties op willekeurige wijze door elkaar worden gebruikt denkt al snel bij zichzelf: zoveel informatie zal ik nooit kunnen onthouden.
In de praktijk valt het echter allemaal wel mee. Het meest belangrijke bij het programmeren is dat men de grondbeginselen begrijpt: het werken met bits en bytes, het werken met vlaggen (dus het geven van signalen) en het kreëren van variabelen in het geheugen. Daar hebben wij bij het bespreken van de Z80-instrukties dan ook de nadruk op gelegd.

Wanneer men zich die grondbeginselen enigermate heeft eigen gemaakt, zal het volgende hoofdstuk, waarin machinetaal-ekwivalenten van Basic-instrukties worden beschreven geen onoverkomelijke problemen opleveren.

Het nadeel van een onderverdeling in hoofdgroepen blijft altijd dat er uitzonderingen voorkomen die niet binnen een van die groepen vallen. We bespreken er hieronder enkele.

a. De RET (return)-instruktie
   Return-instrukties worden gebruikt om vanuit een machinetaal-programma terug te keren naar de Basic-mode, of om het eind van een subroutine aan te geven. Deze instrukties kunnen echter ook aan een voorwaarde worden gebonden. Daarbij worden vlaggen in het vlag-register getest (zie figuur 1.5c) en op grond van de vlag-positie wordt wel of niet teruggekeerd naar een hoofdroutine (of naar de Basic-mode).
   Het betreft de zero-vlag (Z/NZ), de carry-vlag (C/NC), de teken-vlag (M/P) en de pariteits- of overflow-vlag (PE/PO).

b. De NOP (no operation)-instruktie
   Deze instruktie kan gebruikt worden om binnen een programma een ruimte te kreeren die later met bepaalde routines of gegevens moet worden opgevuld.
   De instruktie beïnvloedt een programma niet; de processor doorloopt het gebied tot een werk-opdracht wordt gevonden.

c. De HALT-instruktie
   De HALT-instruktie stopt alle handelingen van de processor, tot een interrupt optreedt.

d. DAA (decimal adjust akkumulator)
   Deze instruktie wordt gebruikt bij het rekenen volgens de BCD-methode (zie hoofdstuk 1.10).
   Volgens de BCD-methode worden de decimale grondgetallen (0 tot 9) opgeslagen in groepjes van 4 bits. Wanneer men nu twee getallen gaat optellen (of aftrekken) dan kan een situatie ontstaan, waarin het resulterende getal in een nibble groter is dan 9. In zo'n geval zorgt de DAA-instruktie ervoor dat het te grote getal met tien wordt verminderd.

Voorbeeld:

```
10 LD A,&H15 ;binair 0001 0101
20 LD B,&H16 ;binair 0001 0110
30 ADD A,B
```

De ADD-bewerking (optellen) zal het volgende binaire resultaat geven: 0010 1011; dat is hexadecimaal &H2B. We zien dat het tweede bit-groepje een getal bevat dat niet tot de decimale grondgetallen 0 tot 9 behoort (1011 = 12). De uitkomst van de optelsom is dus onjuist.
Voegen we nu de DAA-instruktie toe, dan zal de inhoud van de akku worden 'aangepast' en het volgende binaire resultaat ontstaat: 0011 0001; dat is hexadecimaal &H31.
De inhoud van het bit-groepje is met tien verminderd; de waarde 10 is toegevoegd aan het eerste bit-groepje. 15 + 16 geeft nu als som de waarde 31; een korrekt resultaat!

***Let op:***
de hexadecimale notatie geeft bij deze rekenmethode de decimale inhoud van een byte weer!

# 3. Basic-instrukties worden vertaald

## *3.1 De afstand overbrugd*

In de vorige hoofdstukken hebben we ons beziggehouden met de grond beginselen van 'machinetaal-programmering'. Omdat het de bedoeling is dat deze elementaire kennis wordt vertaald in programma's die van praktisch nut voor ons zijn reiken we de beginnende programmeur in dit hoofdstuk een aantal hulp-routines aan, die als grondslag kunnen dienen voor toepassingsprogramma's.

We hebben daarbij gekozen voor een opzet die zoveel mogelijk het verband laat zien tussen machinetaal en de Basic-programmeertaal. Op die manier kan de afstand tussen Basic en machinetaal (juister gezegd: assembler-taal) worden overbrugd en verliest het begrip 'machinetaal' zijn wat duistere, ontoegankelijke klank.

Een dertigtal Basic-kommando's en funkties worden in alfabetische volgorde behandeld. Eerst wordt een korte omschrijving gegeven van de Basic-instruktie (eventueel voorzien van verklarende vorbeelden); daarna worden de overeenkomstige standaard-routines en/of systeem-variabelen (veelal opgenomen in een kort programma) besproken, terwijl in sommige gevallen wordt verwezen naar uitgebreide test-programma's in het slothoofdstuk van het boek. Op die manier ontstaat een uiterst handig en toegankelijk naslagwerk, dat geraadpleegd kan worden bij het zelf schrijven van machinetaal-routines.

***Let op:***
Wanneer we in de hiernavolgende hoofdstukken spreken over 'machinetaal-ekwivalent', dan verwijzen we in feite naar het assembler-bron-programma, dat door de assembler wordt omgezet in machine-kode!

### *3.2 BEEP*

Basic-kommando, waarmee een piep-toon kan worden geproduceerd. Klankkleur en volume van deze toon kunnen alleen op MSX2-machines worden ingesteld (met behulp van het SET BEEP-kommando).

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H00C0 (BEEP), aan te roepen met behulp van de Z80-instruktie CALL. De routine wijzigt de inhouden van de registers AF,BC,DE en maakt de interrupts weer mogelijk: EI.

### *3.3 CHR$ (getal)*

Basic-funktie, waarmee (meestal in samenwerking met het kommando PRINT) ASCII-kode-getallen worden omgezet in tekens, grafische symbolen of opdrachten. Zo plaatst het kommando PRINT CHR$(65) de hoofdletter 'A' op het scherm, terwijl het kommando PRINT CHR$(12) het scherm schoonveegt.

Machinetaal-ekwivalent:

De BIOS-ROM-routines die gegevens uitvoeren naar het scherm of naar de printer (zie ook het Basic-kommando PRINT):

*a.* standaard-routine &H0018 (OUTDO)
het ASCII-kode getal moet in de akku (het A-register) worden geplaatst, waarna de routine wordt aangeroepen met behulp van de instruktie: RST &H18.

| Voorbeelden: | LD A,12 | |
|---|---|---|
| | RST &H18 | (wist het scherm) |
| | LD A,7 | |
| | RST &H18 | (geeft een BEEP-toon) |

Deze routine voert de gegevens altijd uit naar het beeldscherm, behalve als het getal 1 wordt geplaatst in de systeem-variabele &HF416 (PRTFLG); in dat geval zal er een uitvoer naar de printer

plaatsvinden.
De opdracht:

```
LD A,12
RST &H18
```

zal nu een papierdoorvoer veroorzaken!

*b.* standaard-routine &H00A2 (CHPUT)
geeft alleen een uitvoer naar het scherm. De ASCII-kode van het af te drukken teken wordt in register A verwacht. De routine maakt interrupts mogelijk: EI.

*c.* standaard-routine &H00A5 (LPTOUT)
geeft alleen een uitvoer naar de printer. De ASCII-kode van het te printen teken wordt in register A verwacht. De routine wijzigt AF.

### *3.4 CLS*

Basic-kommando, waarmee het scherm in iedere mode kan worden schoongeveegd.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H00C3 (CLS)
deze routine zal alleen werken wanneer de zero-vlag (zie figuur 1.5c) hoog (1) is. Het hoog-maken van de zero-vlag kan men bewerkstelligen door aan de wis-opdracht 'CALL &H00C3' de instruktie 'SUB A' vooraf te laten gaan. Deze instruktie geeft de akku de inhoud 'nul', immers: A wordt van A afgetrokken.

De routine wijzigt: AF, BC, DE. EI.

### *3.5 COLOR voorgrond,achtergrond,rand*

Basic-kommando, waarmee de diverse kleurwaarden van het scherm (en de tekens op het scherm = voorgrond) kunnen worden ingesteld.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H0062 (CHCLR)
deze routine wijzigt de kleuren wanneer voldaan is aan de volgende voorwaarde: de gewenste kleurwaarden moeten van te voren (met behulp van de Z80-load-instruktie) in een aantal systeem-variabelen worden geladen.

Die systeem-variabelen zijn:

a. &HF3E9 (FORCLR) (voorgrond-kleur)
b. &HF3EA (BAKCLR) (achtergrond-kleur)
c. &HF3EB (BORCLR) (border- of rand-kleur).

Voorbeeld:

```
10 ' LD A,15        ;kleurkode voor de kleur 'wit'
20 ' LD (&HF3E9),A ;plaats de kleurkode in FORCLR
30 ' CALL &H0062   ;wijzig de kleur
40 ' RET           ;return - terug naar Basic
50 ' EIND END
```

Deze routine zal de voorgrond-kleur (dus de kleur van de tekens op het scherm) wijzigen. (Was de kleur reeds wit dan blijft de kleur natuurlijk wit...)

De standaard-routine (&H0062) wijzigt: AF,BC,DE,EI.

### *3.6 COPY (kopieer)*

MSX2-Basic-kommando, waarmee de scherm-inhoud (of een gedeelte daarvan) kan worden verplaatst.

Machinetaal-ekwivalent:

Op alle MSX-computers is het kopiëeren van de scherm-inhoud mogelijk met behulp van die BIOS-ROM-routines, die blokken gegevens kunnen verplaatsen van het video-RAM naar het RAM en omgekeerd:

a. standaard-routine èH0059 (LDIRMV):
   kopieer blok VRAM naar het RAM-geheugen.

Het begin-adres van het te verplaatsen blok in het video-RAM moet in registerpaar HL worden geplaatst. Het doel-adres in het RAM-geheugen wordt in registerpaar DE verwacht. Registerpaar BC zal de lengte van het te verplaatsen blok moeten bevatten.

De routine wijzigt: AF, BC, DE, EI.

b. standaard-routine èH005C (LDIRVM): kopieer blok gegevens van het RAM-geheugen naar bet VRAM.

Het start-adres in RAM wordt in registerpaar HL geplaatst. Het doel-adres in het video-RAM wordt in registerpaar DE geplaatst. De lengte van het te verplaatsen blok wordt in registerpaar BC verwacht.

De routine wijzigt: AF, BC, DE, EI.

Let op: hoofdstuk 6.9 bevat een uitgebreid test-programma, waarin wordt getoond hoe binnen een Basic-programma blokken tekst kunnen worden verplaatst met behulp van een machinetaal-routine.

### *3.7 CSRLIN (cursor-line)*

Basic-instruktie, waarmee het regelnummer waarop de kursor zich bevindt kan worden doorgegeven aan een variabele.

Voorbeeld:

Y=CRSLIN variabele Y bevat het regelnummer, waarnaar kan worden teruggesprongen (indien gewenst) met behulp van het Basic-kommando LOCATE (bijvoorbeeld LOCATE 0,Y).

Machinetaal-ekwivalent:

Vanuit een machinetaal-programma kan informatie over de positie van de kursor worden ontleend aan de volgende systeemvariabelen:

&HF3DC (CSRY)

bevat het regelnummer van de kursor (wanneer de computer zich in de tekst-mode bevindt);

&Hf3DD (CSRX)

bevat het kolom-nummer van de kursor (wanneer de computer zich in tekst-mode bevindt).

Wil men de positie van de kursor beïnvloeden, dan kan men dat bewerkstelligen door de gewenste waarden in deze variabelen te plaatsen met behulp van de Z80-load-instruktie.

Voorbeeld:

```
10 ' LD A,10
20 ' LD (&HF3DC),A ; laad regelnummer 10 in CSRY
```

### *3.8 ERL (error-line)*

Variabele waarin door de foutverwerkingsroutine van het Basic-vertaal-programma (de interpreter) het nummer van de regel waarin een fout wordt ontdekt wordt opgeslagen.

Machinetaal-ekwivalent:

Vanuit een machinetaal-programma kan het nummer van de regel waarin een fout is opgetreden worden opgevraagd door het uitlezen van de systeemvariabele &HF6B3 (ERRLIN).

Voorbeeld:

```
LD A, (&HF6B3)
```

### *3.9 FOR ... NEXT*

Basic-kommando, waarmee een opdracht (ook de opdracht om niets te doen) meerdere malen kan worden herhaald.

Voorbeeld:

```
FOR C=0 TO 10:PRINT C:NEXT C
```

De variabele C wordt in dit geval geladen met het startgetal 0 en nadat de opdracht 'PRINT C' is uitgevoerd wordt de variabele C (die hier als teller fungeert) met de waarde 1 verhoogd door het kommando 'NEXT', net zolang tot de teller C de waarde 10 heeft bereikt; op dat moment stopt de routine.

Het programma-gedeelte tussen de 'FOR'-instruktie en de 'NEXT'-instruktie noemen we een 'programma-lus'. Een programma-lus heeft de volgende kenmerken:

a. een variabele wordt als teller gebruikt;

b. er moet teruggesprongen kunnen worden;

c. wil men uit de lus springen, dan zal er een voorwaardelijke instruktie aanwezig moeten zijn.

De FOR-NEXT-programma-lus in ons voorbeeld voldoet aan alle voorwaarden. Er is een teller: de variabele C. Er is een sprongopdracht: de instruktie 'NEXT'. Er wordt alleen gesprongen als de waarde van C niet gelijk is aan 10; is de waarde groter dan wordt uit de lus gesprongen.

Een soortgelijke konstruktie kunnen we met behulp van een aantal eenvoudige Basic-instrukties zelf ontwerpen:

Voorbeeld:

```
10 C=0:UIT=10
20 PRINT C
30 C=C + 1:IF C <> UIT THEN 20
```

We definiëren in regel 10 de teller C en de eind-waarde UIT. In regel 30 wordt de teller verhoogd (C=C+1) en vergeleken met de waarde in variabele UIT.

Alleen als de uitkomst van de vergelijking 'waar' is (dus als C niet

gelijk is aan UIT) zal worden teruggesprongen naar regel 20. In het andere geval (C is gelijk aan UIT) zal het programma worden voortgezet; de lus is 'opengebroken'.

Machinetaal-ekwivalent:

In machinetaal-programma's wordt een programma-lus opgebouwd volgens het principe dat hierboven werd omschreven: we moeten variabelen in het leven roepen die als 'teller' en als 'eind'- of referentie-waarde dienst kunnen doen (bijvoorbeeld een Z80-register of een lokatie in het RAM-geheugen). We moeten de variabele die als teller fungeert verhogen (of verlagen) en we moeten een procedure ontwerpen, die onderzoekt of aan een bepaalde (door ons gestelde) voorwaarde is voldaan.

Als voorbeeld zetten we het kleine Basic-programma dat hierboven staat afgebeeld om in een reeks assembler-instrukties:

Voorbeeld:

```
10 ' ORG 49500
15 ' LD C,0       ;het C-register fungeert als teller
20 ' LD B,10      ;het B-register bevat de UIT-waarde
25 ' ;
30 ' LUS LD A,C ;plaats inhoud C in de akku
35 ' ADD A,48     ;zet getal in C om in ASCII-teken
40 ' RST &H18     ;print het getal m.b.v. routine OUTDO
45 ' INC C        ;verhoog de teller -> C=C+1
50 ' LD A,C       ;plaats inhoud C in de akku
55 ' CP B         ;vergelijk C met de inhoud van B
60 ' JP NZ,LUS    ;spring naar LUS als B <> C
65 ' ;
70 ' RET          ;return - terug naar Basic
75 ' EIND END
```

De regels 30 tot en met 60 vormen de programma-lus. De print-opdracht staat in de regels 30 tot en met 40. De procedure waarin wordt bekeken of er voldaan is aan een bepaalde voorwaarde is te vinden in de regels 50 tot en met 60. De inhoud van register C (de teller) wordt (via register A) vergeleken met de inhoud van register B.

Zijn de waarden van B en C niet gelijk aan elkaar dan zal de zero-vlag laag (0) zijn, d.w.z. er bestaat een NZ-(not-zero)-toestand; het gevolg is dat vanuit regel 60 wordt terugggesprongen naar het begin van de LUS in regel 30. Zodra de inhouden van B en C gelijk zijn aan elkaar wordt de zero-vlag hooggemaakt; er ontstaat een Z-(zero)-toestand, hetgeen betekent dat er uit de LUS gesprongen wordt. Via de instruktie 'RET'-urn in regel 70 wordt teruggesprongen naar de Basic-mode.

Er zijn verschillende mogelijkheden denkbaar om programma-lussen te konstrueren binnen een machinetaal-programma. De figuren 3.9a en 3.9b laten een aantal van die mogelijkheden zien.

***Let op:***
Onderdeel *d* van figuur 3.9a toont ons een instruktie die in het hoofdstuk over sprong-instrukties (2.3) wel werd vermeld maar niet expliciet besproken: DJNZ (Decrement - Jump - Not Zero). Dit is een instruktie die het Basic-FOR-NEXT-kommando beel dicbt benadert.
Register B fungeert hier als teller en bevat tevens de eindwaarde. Wanneer we de gewenste eindwaarde in dit register bebben geplaatst zorgt de DJNZ-instruktie er voor dat er net zolang wordt teruggesprongen naar het begin van de LUS totdat de inhoud van register B gelijk is aan nul.

```
----------------------------------------------------------------
Machinetaal-lussen m.b.v. 8-bits registers
----------------------------------------------------------------
a. LD r,n                               r  = A,B,C,D,E,H,L
   LUS PUSH rr ;bewaar rr               rr = AF,BC,DE,HL
   machinetaal-routine                  n  = getal (0-255)
   POP rr      ;herstel rr
   DEC r       ;r=r-1                   verlaag teller r
   JP NZ,LUS                            IF r < > 0 THEN LUS
----------------------------------------------------------------
b. LD r,0                               r : zie boven
   LUS PUSH rr                          rr: zie boven
   machinetaal-routine
   POP rr
   INC r       ;r=r+1                   verhoog teller r
   CP n        ;n=getal                 vergelijk met getal
   JP NZ,LUS                            IF r < > n THEN LUS
----------------------------------------------------------------
c. LD A,n      ;n=getal                 A = akku
   LUS EX AF,AF'                        AF in hulpregisters
   machinetaal-routine
   EX AF,AF'                            herstel inhoud AF
   DEC A       ;A=A-1                   verlaag teller A
   JP NZ,LUS                            IF A < > 0 THEN LUS
----------------------------------------------------------------
d. LD B,n      ;n=getal
   LUS PUSH BC                          bewaar inhoud B
   machinetaal-routine
   POP BC                               herstel inhoud B
   DJNZ LUS    ;B=B-1                   IF B <> 0 THEN LUS
----------------------------------------------------------------
```
*figuur 3.9a*

```
-----------------------------------------------------------
Machinetaal-lussen m.b.v 16-bits registerparen
-----------------------------------------------------------
a. LD rr,nn                         rr = BC,DE,HL
   LUS PUSH rr ;bewaar rr           nn = getal (0-65535)
   machinetaal-routine
   POP rr       ;herstel rr
   DEC rr       ;rr=rr-1            DEC-instruktie
   LD A,hr      ;hoge helft rr      beinvloedt vlaggen niet
   OR lr        ;lage helft rr      vandaar de OR-bewerking
   JP NZ,LUS                        IF (hr OR lr)
                                    <> 0 THEN LUS
-----------------------------------------------------------
b. LD rr,0                          rr: zie boven
   LUS PUSH rr ;bewaar rr
   machinetaal-routine
   POP rr       ;herstel rr
   INC rr       ;rr=rr+1            verhoog de teller
   LD A,hr      ;hoge helft rr      vergelijk inhoud hr met
   CP n         ;getal 1            getal 1 (0-255)
   JP NZ,LUS                        IF hr <> n THEN LUS
   LD A,lr      ;lage helft rr      als hr=n vergelijk dan
   CP n         ;getal 2            inhoud lr met getal 2
   JP NZ,LUS                        IF lr <> n THEN LUS
-----------------------------------------------------------
```

*figuur 3.9b*

### *3.10 GOSUB regelnummer (Goto SUBroutine)*

Basic-kommando, waarmee binnen het programma tijdelijk gesprongen kan worden naar een ander deel van het programma: de sub-routine. Die sub-routine moet altijd beeindigd worden met het Basic-kommando 'RETURN': de opdracht om terug te keren naar het hoofdprogramma.

Om de terugkeer vanuit een sub-routine naar het hoofdprogramma mogelijk te maken wordt het regelnummer waarin de 'GOSUB'-instruktie zich bevindt (tesamen met een aantal andere waarden, zoals het adres van de instruktie in het geheugen en het 'GOSUB'-token) tijdelijk opgeslagen in het RAM-geheugen (ze

worden op de stapel ge-PUSH-ed). Ontdekt de Basic-interpreter een 'RETURN'-instruktie dan zal dit geheugengebied weer worden uitgelezen en er zal (indien een 'GOSUB'-token werd ontdekt) worden teruggesprongen naar het hoofd-programma.

Voegen we aan de 'RETURN'-instruktie een regelnummer toe (bijvoorbeeld RETURN 20) dan wordt het terugkeer-adres dat bij de 'GOSUB'-instruktie hoort genegeerd en er zal een sprong worden gemaakt naar het begin van de aangegeven regel!

Voorbeeld:

```
10 GOSUB 40:PRINT "GEEN REGEL-NUMMER":END
20 PRINT "REGEL-NUMMER TOEGEVOEGD"
30 END
40 RETURN 20
```

Uitvoering van dit test-programma zal als resultaat de tekst:

```
"REGEL-NUMMER TOEGEVOEGD"
```

op het scherm plaatsen. Er werd teruggesprongen naar regel 20.

Machinetaal-ekwivalent:

De Z80-Instruktie: CALL adres
Wordt in een Basic-programma gesprongen naar een programmaregel (bijvoorbeeld GOSUB 40), in een machinetaal-programma worden sprongen gemaakt naar lokaties in het geheugen, die worden aangeduid met 'adressen' (bijvoorbeeld CALL &H00C0 - roep de subroutine op adres &H00C0 aan).

Een machinetaal-subroutine moet, zoals de Basic-programmeertaal dat ook vereist, worden afgesloten met een Return-opdracht: De Z80- instruktie 'RET'.

Willen we zelf bepalen naar welk adres moet worden teruggesprongen dan zullen we het terugkeer-adres dat door de Z80-processor in het stapel-geheugengebied werd geplaatst bij het verwerken van de 'CALL'-instruktie moeten wijzigen met behulp van de stapelinstrukties 'POP' (haal een adres van de stapel af) en 'PUSH' (zet

een adres op de stapel).

Voorbeeld:

```
10 ' ORG 49500
15 ' CALL TEST     ;roep subroutine TEST aan
20 ' DEEL1 LD A,49 ;plaats ASCII-waarde getal 1 in A
25 ' RST &H18      ;print het getal m.b.v. routine OUTDO
30 ' RET           ;keer terug naar Basic
35 ' ;
40 ' DEEL2 LD A,50 ;plaats ASCII-waarde getal 2 in A
45 ' RST &H18      ;print het getal
50 ' RET           ;keer terug naar Basic
55 ' ;
60 ' TEST POP HL   ;haal terugkeeradres van de stapel
65 ' LD HL,DEEL2   ;plaats startadres DEEL2 in HL
70 ' PUSH HL       ;zet dit adres op de stapel
75 ' RET           ;haal terugkeeradres van de stapel en
80 ' ;              maak een sprong naar dat adres
85 ' EIND END
```

Wanneer we dit programma uitvoeren (nadat het met behulp van de assembler is vertaald) dan zullen we kunnen konstateren dat het getal 2 op het scherm wordt afgedrukt. Dat betekent dat niet de print-opdracht in de hoofd-routine (regels 15 tot 30) werd uitgevoerd, maar de print-opdracht in routine DEEL2 (regels 40 tot 50).
We kunnen daaruit de konklusie trekken dat de subroutine TEST inderdaad het terugkeeradres gewijzigd heeft. Tevens kunnen we konkluderen dat het gebruik van 'POP'- en 'PUSH'-instrukties in een subroutine een riskante zaak is; een slordig gebruik van deze instrukties zal het terugkeeradres aantasten en op die manier het vastlopen van de machine bewerkstelligen!

### *3.11 GOTO regelnummer*

Basic-kommando, waarmee een sprong kan worden gemaakt naar een willekeurige Basic-programma-regel.

Machinetaal-ekwivalent:

De Z80-spong-instrukties:

J*um*P adres
J*um*P (HL) - spring naar een adres dat in registerpaar HL staat

Er wordt altijd gesprongen naar een lokatie in het beschikbare geheugen (adres 0 - 65535). Er kan naar een willekeurig adres worden gesprongen, maar in het geval van een ontoelaatbare sprong zal geen foutmelding volgen; in veel gevallen loopt de machine vast waardoor de kans bestaat dat het ingetypte programma verloren gaat.

Zie voor een ontsnappingsmogelijkheid hoofdstuk 2.2c.
Zie voor sprong-instrukties hoofdstuk 2.3.

### *3.12 IF ... THEN (als - dan)*

Basic-kommando, waarvan de betekenis als volgt kan worden omschreven:

*ALS* aan een bepaalde voorwaarde wordt voldaan, *DAN* moet een opdracht worden uitgevoerd.

Voorbeeld:

```
IF A=0 THEN PRINT "0"
```

Machinetaal-ekwivalent:

De voorwaardelijke (aan een voorwaarde gebonden) Z80-instrukties JumP, Jump Relative, CALL en RETurn:

a. JP/CALL Z,adres — JP/CALL NZ,adres
   JP/CALL C,adres — JP/CALL NC,adres
   JP/CALL M,adres — JP/CALL P,adres
   JP/CALL PE,adres — JP/CALL PO,adres

b. JR Z,n — JR NZ,n
   JR C,n — JR NC,n
   DJNZ n

c. RET Z — RET NZ
RET C — RET NC
RET M — RET P
RET PE — RET PO

Al deze instrukties zijn gekoppeld aan een voorwaarde. Zo is de betekenis van de opdracht 'JP Z,adres' omgezet in Basic-uitdrukkingen:
IF zero-vlag = 1 THEN GOTO adres.

De betekenis van de opdracht 'CALL NZ,adres' is:
IF zero-vlag = 0 THEN GOSUB adres.

De betekenis van RET M is:
IF tekenvlag = 1 THEN RETURN (zie figuur 1.5c).

We zien dat in machinetaal alleen voorwaardelijke sprong-opdrachten bestaan (de return-instruktie doet in feite ook niets anders dan terugspringen naar een bepaald adres).

In Basic kunnen we de computer de opdracht

```
IF A=0 THEN PRINT"0"
```

geven; in een machinetaal-programma zullen we met behulp van een sprong-instruktie moeten verwijzen naar een routine die een bepaalde opdracht uitvoert.

Dat is het grote verschil tussen Basic en machinetaal: veel zaken die in Basic vanzelfsprekend zijn, zijn dat helemaal niet meer wanneer er wordt geprogrammeerd in machinetaal (juister gezegd: assembler-taal). De computer gedraagt zich ineens als een klein, onmondig kind en de programmeur is de onderwijzer die dat lastige, domme kind haarfijn alles uit moet leggen om het ertoe te bewegen een moeilijke opdracht uit te voeren. Dat betekent dat ieder probleem ontleed moet worden in kleine deel-probleempjes; dat iedere moeilijke opgave teruggebracht moet worden tot een reeks eenvoudige opdrachten.

In feite maakt dat het programmeren in machinetaal zo ingewikkeld: men is gedwongen met grote aantallen instrukties te werken, zodat

het overzicht op het geheel al snel verloren gaat. Het is daarom belangrijk een machinetaal-programma op te bouwen uit blokken (of modules), die ieder afzonderlijk kunnen worden getest. Wanneer de routines goed blijken te werken kunnen ze tot een geheel aaneen worden gesmeed.

Ook in het geval van het konstrueren van een 'IF-THEN'-routine is het belangrijk om eerst te bekijken uit welke onderdelen een dergelijke konstruktie moet bestaan. We stellen onszelf daartoe de hele simpele vraag: Wat hebben we nodig? Het antwoord is niet zo moeilijk te geven:

a. de hierbovengenoemde voorwaardelijke Z80-instrukties, die samenwerken met het vlag-register;

b. instrukties die de vlaggen (zero, carry, teken, pariteit/overflow) beïnvloeden, met andere woorden: instrukties die voorwaarden scheppen.

Laten we nog eens kijken naar het kleine Basic-voorbeeld-programma:

```
IF A=0 THEN PRINT "0"
```

We zien dat de Basic-regel de opdracht bevat om de inhoud van de variabele A te vergelijken met de konstante 'nul'. Wel, die opdracht kunnen we heel eenvoudig omzetten in machinetaal, omdat de Z80-instruktie-set de vergelijkings-opdracht 'Compare' bevat (zie hoofdstuk 2.9c).

In dit geval moeten we de instruktie 'CP 0' gebruiken (dat wil zeggen: vergelijk de inhoud van register A met de konstante 0). De Z80-processor zet het resultaat van deze logische bewerking in het vlag-register: de zero-vlag wordt hoog gemaakt bij gelijke getallen; we schrijven 'Z' (zero).
De zero-vlag wordt laag gemaakt wanneer de getallen niet gelijk zijn aan elkaar; we schrijven 'NZ' (not-zero). De andere vlaggen interesseren ons nu niet omdat we alleen maar willen weten of A gelijk is aan 'nul'.

In het Basic-programma laten we het testen van de vlaggen over aan

de 'interpreter'; die is het hulpje, dat het vervelende werk voor ons moet opknappen.

Een machinetaal-programmeur echter is een zeer sociaal mens: hij delegeert niet, hij doet het vervelende werk zelf. Hij krabt zich eens achter de oren en vraagt zichzelf af hoe hij de computer duidelijk kan maken dat na de vergelijkings-instruktie een print-opdracht moet worden uitgevoerd. Hij bekijkt de lijst van voorwaardelijke instrukties en kiest daaruit de instruktie 'RET NZ', een instruktie die een return-opdracht uitvoert als de zero-vlag laag is (A <> 0).

Met behulp van die instruktie kan het Basic-voorbeeld-programma als volgt worden vertaald:

```
10 ' CP 0          ;vergelijk inhoud A met 'nul'
20 ' RET NZ        ;IF A <> 0 THEN RETURN
30 ' LD A,48       ;plaats ASCII-waarde getal 0 in A
40 ' RST &H18      ;print het getal 0.
50 ' RET           ;return - keer terug naar Basic
60 ' EIND END
```

De Basic-regel 'IF A=0 THEN PRINT "0"' is omgezet in een assembler-bronprogramma, dat door de assembler vertaald kan worden in machinekode.

### *3.13 INP poortnummer*

Basic-funktie, waarmee de poorten van de computer kunnen worden 'uitgelezen'.

Voorbeeld:

```
PRINT INP(&H90)
```

(kijk welke waarde zich in de Printer-Status-Poort bevindt).

Wanneer uitlezen van status-poort &H90 de waarde 255 geeft dan is er of geen printer aangesloten of de printer bevindt zich in een tijdelijke 'stop-mode' (hij staat niet aan). Wanneer de printer vrij is voor het afdrukken van gegevens dan geeft uitlezing van deze poort

de waarde 253.

Op die manier kan een kontrole-routine ingebouwd worden in een Basic-programma, die ervoor zorgt dat alleen dan een print-opdracht aan de computer wordt gegeven als de printer 'schrijf-klaar' is.

Voorbeeld:

```
10 IF INP(&H90)=255 THEN GOSUB 40
20 LPRINT "PRINTER GEREED"
30 END
40 INPUT "PRINTER INGESCHAKELD";A$
50 IF INP(&H90)=253 THEN RETURN
60 GOTO 40
```

In regel 10 wordt de printer-status-poort uitgelezen. Is de inhoud 255 dan wordt de subroutine aangeroepen, die de gebruiker vraagt of de printer ingeschakeld is. Omdat in regel 50 de status-poort opnieuw wordt uitgelezen zal een terugkeer uit de routine alleen mogelijk zijn wanneer de printer wordt ingeschakeld (of aangesloten).

***Let op:***
Het is mogelijk dat het uitlezen van de printerpoort een andere waarde oplevert. In dat geval geldt de volgende regel: wanneer bit 1 hoog is dan is de printer *niet* ingeschakeld (zie in het voorbeeld de waarde 255); is bit 1 laag, dan is de printer gereed voor gebruik (zie in het voorbeeld de waarde 253). De waarden 255 en 253 moeten nu door de juiste waarden worden vervangen!

Machinetaal-ekwivalent:

De Basic-funktie 'INP' verwijst naar de Z80-IN-instruktie (zie hoofdstuk 2.19).
Deze Z80-instruktie heeft de volgende vorm: IN reg,(C) - waarbij register C het poort-nummer moet bevatten. Omdat we een test-procedure willen ontwerpen kiezen we als inlees-register het Reken-register van de Z80-processor: de akku of register A, omdat dit register ons de meeste mogelijkheden biedt.

We konstrueren nu de volgende test-routine (die wat eenvoudiger is dan de Basic-test-routine, maar even effektief):

```
10 ' ORG 49500
20 ' LUS LD C,&H90      ;plaats poort-nummer in register C
30 ' IN A,(C)           ;plaats poort-inhoud in register A
40 ' CP 253             ;is de printer schrijfklaar?
50 ' RET Z              ;ja, dan return - naar Basic
60 ' CALL &H00C0        ;nee, dan BEEP
70 ' JP LUS             ;terug naar begin van de routine
80 ' EIND END
```

Wanneer we deze routine met behulp van de assembler uitvoeren, dan zal er niets gebeuren als de printer 'schrijfklaar' is (dus aangesloten en ingeschakeld). In het andere geval zal er een aanhoudende pieptoon klinken, die pas ophoudt wanneer de printer wordt ingeschakeld.

Omdat het direkt werken met de poorten van de MSX-computer wordt afgeraden voor algemeen gebruik, bevat het MSX-BIOS-ROM een routine die de printer-statuspoort uitleest en de inhoud test: de standaardroutine `&H00A8 (LPTSTT)`.
Wanneer de printer schrijfklaar is wordt de zero-vlag laag gemaakt (NZ); in het andere geval is de zero-vlag hoog (Z).

De test-routine ziet er als volgt uit, wanneer we gebruik maken van deze ROM-routine:

```
10 ' ORG 49500
20 ' LUS CALL &H00A8 ;lees poort &H90 uit
30 ' RET NZ          ;return als printer schrijfklaar is
40 ' CALL &H00C0     ;BEEP in het andere geval
50 ' JP LUS          ;spring terug naar begin van de lus
60 ' EIND END
```

Het spreekt vanzelf dat dit programma na uitvoering hetzelfde resultaat zal geven.

### *3.14 INPUT$(x)*

Basic-funktie, waarmee de uitvoering van een programma onderbroken kan worden. Pas na het intoetsen van x aantal tekens zal het programma worden voortgezet.

Voorbeeld:

```
10 A$=INPUT$(1)
20 IF A$="1" THEN PRINT A$
30 GOTO 10
```

Zolang er geen toets wordt ingedrukt zal de computer wachten. Na het indrukken van een toets wordt het gekozen teken doorgegeven aan de string-variabele A$, waarvan de inhoud in regel 20 wordt vergeleken met de konstante waarde "1". Wanneer de vergelijking een 'onwaar' resultaat oplevert (A$ < > "1") dan wordt via regel 30 teruggesprongen naar regel 10: Er is een eindeloze lus ontstaan, die men alleen kan verlaten na het intoetsen van het juiste karakter.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H009F (CHGET), aan te roepen met de Z80-instruktie CALL.

Deze routine heeft ongeveer hetzelfde effekt als de Basic-funktie 'INPUT$(1)'.
Er wordt gekontroleerd of er een toets wordt ingedrukt en wanneer dat niet het geval is wordt de uitvoering van het machinetaal-programma stopgezet. Na het indrukken van een willekeurige toets zal de nitvoering van het programma worden voortgezet, tenzij we deze standaard-routine kombineren met een test-programma dat onderzoekt welke toets er wordt ingedrukt, zoals we dat hierboven hebben gedaan in het Basic-voorbeeld-programma.

Voorbeeld:

```
10 ' ORG 49500
20 ' LUS CALL &H009F ;roep routine CHGET aan
30 ' CP 49          ;vergelijk inhoud A met ASCII-waarde 1
40 ' JP NZ,LUS      ;IF A <> 49 THEN LUS
50 ' RST &H18       ;print getal 1
60 ' RET            ;return - terug naar Basic
70 ' EIND END
```

In regel 30 wordt een vergelijkingsopdracht ('CP 49') aan de computer gegeven. Dat is mogelijk omdat de routine CHGET de ASCII-

waarde van de ingedrukte toets in register A plaatst (het zelf kiezen van een variabele is hier dus niet nodig).

De routine CHGET schakelt de interruptverwerker in.

### *3.15 INTERVAL*

Basic-kommando, dat in kombinatie met het kommando 'ON INTERVAL=x GOSUB' de mogelijkheid biedt het programma met tussenpozen van x maal 1/50e sekonde te onderbreken, teneinde een aantal opdrachten uit te voeren die liggen besloten in een subroutine.

Voorbeeld:

```
10 SCREEN 1:COLOR 1,5,1
15 FOR I=0 TO 7
20 A$=A$ + CHR$(255):NEXT I
25 SPRITE$=A$:X=56
30 INTERVAL ON
35 ON INTERVAL=50 GOSUB 55
40 CLS:LOCATE 6,10
45 PRINT "INTERVAL-TEST"
50 GOTO 50
55 PUT SPRITE 0,(X,78),15,1
60 X=X+8:IF X>180 THEN X=56
65 RETURN
70 END
```

Dit test-programma laat een blokvormige sprite (gedefinieerd in de regels 15 - 25) heen en weer bewegen over het scherm. De beweging wordt bepaald in de regels 55 - 60: De variabele X wordt iedere sekonde (50 maal 1/50 = 1) met de waarde 8 verhoogd; dat betekent dat de sprite bij elke beweging een letter van de tekst "INTERVAL-TEST" overlapt. Wanneer X groter is dan 180 wordt teruggesprongen naar de startwaarde.

Het bijzondere van dit programma is dat de beweging automatisch verloopt. We hoeven dus geen programma-lus te bouwen, die de beweging van de sprite regelt.

Machinetaal-ekwivalent:

De interrupt-routine op adres &H0038 (KEYINT), die 50 maal per sekonde wordt aangeroepen door de video-processor. Deze routine wordt niet rechtstreeks aangeroepen, maar indirekt, via de 'HOOK' op adres &HFD9F (HTIMI).

Een 'hook' is een reeks van 5 bytes, die in het algemeen gevuld zijn met de waarde 201 (het kodegetal van de Z80-instruktie 'RET'). De MSX-computer kent 112 hook-adressen, die zich bevinden in het geheugengebied &HFD9A tot &HFFC5.

Door de geheugen-ruimte van 5 bytes op te vullen met een 'CALL'-opdracht bezitten we de mogelijkheid een zelfgeschreven programma in werking te stellen, iedere keer wanneer de betreffende 'hook' wordt aangeroepen vanuit een Basic-routine of vanuit een BIOS-ROM-routine.

***Let op:***
Voordat men overgaat tot het aanroepen van een 'hook', dient men eerst te kontroleren of de 'hook' al dan niet in gebruik is! De 'hook' zal vrij zijn als het eerste byte de waarde 201 (return-kode) bevat.
Wanneer men de beschikking heeft over het programma-pakket FLASH (assembler en dis-assembler), dan kan men het hook-gebied uitlezen door het intoetsen van de bovengenoemde start- en eindwaarden en de listing met behulp van de printer op papier zetten. Een lijst van hook-adressen is te vinden in het 'MSX-ROM-BIOS-Handboek' en het 'MSX2 Zakboekje' - zie hoofdstuk 2.16b.

In hoofdstuk 6.10 is een testprogramma te vinden, waarin wordt getoond hoe men gebruik kan maken van de interrupt-verwerker van de MSX-computer.

## *3.16 KEY ON/OFF*

Basic-kommando, waarmee de tekst van de funktie-toetsen op het scherm kan worden geplaatst, respektievelijk verwijderd.

Machinetaal-ekwivalent:

De BIOS-ROM-routines:

&HOOCF (DSPFNK)- tekst op scherm
wijzigt: AF,BC,DE,EI

&H00CC (ERAFNK)- tekst verwijderen
wijzigt: AF,DE,EI

Voorbeeld:

```
10 ' ORG 49500
20 ' CALL &H00CC
30 ' RET ;naar Basic
40 ' EIND END
```

Na uitvoering van deze routine zal de tekst van de funktietoetsen verdwenen zijn (we hebben het kommando 'KEY OFF' ontworpen).

### *3.17 LINEINPUT ("tekst") string-variabele*

Basic-kommando, waarmee een gehele tekst-regel (inklusief komma's en aanhalingstekens) kan worden doorgegeven aan een string-variabele.

Voorbeeld:

```
10 LINEINPUT A$
20 PRINT A$
```

Wanneer we dit programma RUNnen zal de kursor in beeld verschijnen en er zal worden gewacht op het intoetsen van tekst. Na het indrukken van de ENTER-toets wordt de tekst-invoer beëindigd en kan het programma worden voortgezet. Typen we getallen in, dan zullen die als string-variabele worden opgeslagen in het geheugen. Het is derhalve niet mogelijk met dergelijke getallen berekeningen uit te voeren.
Willen we de ingevoerde getallen toch gebruiken in een rekenprogramma, dan zullen we eerst de string moeten omzetten in een numerieke variabele met behulp van de Basic-funktie 'VAL' [b.v. A=VAL(A$)].

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H00B1 (INLIN)

Deze standaard-routine plaatst de kursor op het scherm en wacht op de invoer van tekst, zoals dat in het bovenstaande Basic-programma ook gebeurt.
Willen we eerst een toelichtende tekst op het scherm plaatsen, dan zal er een PRINT-opdracht aan de routine INLIN vooraf moeten gaan.

Voorbeeld:

```
10 ' ORG 49500
15 ' LD A,63 ;ASCII-waarde van ?
20 ' RST &H18 ;print het teken ?
25 ' CALL &H00B1 ;roep routine INLIN aan
```

De vraag is nu: waar in het geheugen plaatst de routine INLIN de ingevoerde tekst?
Wanneer we in Basic programmeren hoeven we die vraag niet te stellen. De ingevoerde tekst wordt in het bovenstaande voorbeeld automatisch toegekend aan de string-variabele A$. We vergeten daarbij vaak dat de Basic-interpreter een groot aantal handelingen moet verrichten om die tekst als string-variabele in het string-geheugen te kunnen plaatsen.

Al die stappen hoeven wij gelukkig niet te zetten. Het grote voordeel van zelf programmeren in machinetaal is dat we een routine kunnen schrijven die is gericht op het doel dat ons voor ogen staat, zonder dat we daarbij rekening hoeven te houden met verschillende andere mogelijkheden.

Het is nu juist de taak van de Basic-interpreter om met zoveel mogelijk faktoren rekening te houden. Dat is zijn kracht, maar tegelijkertijd zijn zwakte, omdat de uitvoeringstijd daardoor aanzienlijk wordt vertraagd. Het is dus niet zo'n ramp dat we wat meer vragen moeten stellen, want de antwoorden stellen ons in staat doeltreffender om te gaan met onze computer.

De standaard-routine INLIN nu plaatst de ingevoerde tekst niet in

het Basic-string-geheugen, maar in de systeem-buffer 'BUF', een gebied in het werkgeheugen van de MSX-computer met het startadres &HF55E. Dat werkgeheugen wordt door de Basic-interpreter voortdurend geraadpleegd en datzelfde zullen wij ook moeten doen. De routine INLIN maakt het ons wat dat betreft gemakkelijk, omdat het adres van de geheugenlokatie die voorafgaat aan de buffer in het registerpaar HL wordt geplaatst.

Willen we het adres van het eerste karakter in de buffer achterhalen dan zal de waarde in HL met 1 verhoogd moeten worden met behulp van de instruktie 'INC HL'.

We kunnen nu vrijelijk beschikken over de ingevoerde tekst en er de bewerkingen mee uitvoeren die voor ons programma noodzakelijk zijn.

Het vervolg van het bovenstaande voorbeeldprogramma zou er als volgt uit kunnen zien:

Voorbeeld:

```
30 ' LUS INC HL         ;HL=HL+1
35 ' LD A,(HL)          ;plaats inhoud 1e karakter in A
40 ' RST &H18           ;print het karakter
45 ' CP 0               ;is het einde van de regel bereikt?
50 ' RET Z              ;ja, dan return - naar Basic
55 ' JP LUS             ;herhaal
60 ' EIND END
```

De ingevoerde tekst wordt teken voor teken op het scherm gezet door de routine in de regels 30 tot 60. Wordt de waarde 'nul' in de regel aangetroffen (een waarde die de routine als eind-markering heeft toegevoegd aan de tekst) dan wordt teruggesprongen naar Basic (regels 45 en 50).

**Let op:**
Er is nog een tweede BIOS-ROM-routine, die een ingetypte regel kan inlezen: de standaard-routine &H00B4 (QINLIN). Deze routine plaatst eerst een vraagteken op het scherm en wacht daarna op tekst-invoer.
Beide routines wijzigen: AF,BC,DE,HL,EI.

### *3.18 LOCATE x,y*

Basic-kommando, waarmee de kursor op elke gewenste positie op het scherm kan worden geplaatst (wanneer de computer zich in de tekst-mode bevindt).

x = kolom-nummer
y = regel-nummer

Machinetaal-ekwivalent:

Binnen een machinetaal-programma kan de positie van de kursor worden beïnvloed door de gewenste x- en y-waarden in de volgende systeem-variabelen te plaatsen:

a. &HF3DD (CSRX) - kolom-nummer
b. &HF3DC (CSRY) - regel-nummer

Voorbeeld:

```
10 ' ORG 49500
20 ' LD A,20           ;kolom-nummer in A
30 ' LD (&HF3DD),A     ;plaats het nummer in CSRX
40 ' LD A,3            ;regel-nummer in A
50 ' LD (&HF3DC),A     ;plaats het nummer in CSRY
60 ' LD A,42           ;ASCII-waarde van het teken '*'
70 ' RST &H18          ;print het teken
80 ' RET               ;return - terug naar Basic
90 ' EIND END
```

Uitvoering van dit programma zal als resultaat een sterretje opleveren, geplaatst op regel 3, kolom 20.

### *3.19 LPRINT*

Basic-kommando, waarmee gegevens naar een printer kunnen worden gestuurd.

Machinetaal-ekwivalent:

De BIOS-ROM-routines:

&H00A5 (LPTOUT), aan te roepen met de instruktie 'CALL' (CALL &H00A5).

De ASCII-waarde van het af te drukken teken wordt in register A verwacht.

De routine wijzigt: AF

&H0018 (OUTDO), aan te roepen met de instruktie 'RST' (RST &H18);

Voorwaarde is dat de waarde 1 geplaatst wordt in de systeem-variabele &HF416 (PRTFLG). De routine maakt interrupts mogelijk.

Zie ook hoofdstuk 3.3 (CHR$).

### *3.20 OUT registernummer,te verzenden waarde*

Basic-kommando, waarmee gegevens getransporteerd kunnen worden naar de randapparatuur van de Z80-processor (zoals het video-RAM, de printer en de geluidsprocessor).

Elke poort heeft een nummer. Hoewel de poort-nummering niet onder de MSX-standaard valt zullen de verschillen in de praktijk gering zijn. Vandaar dat we de input- en output-instrukties in dit boek vrij uitvoerig behandelen.

Willen we bijvoorbeeld met behulp van het OUT-kommando gegevens uitvoeren naar de printer dan zullen we de nummers moeten weten van twee poorten:

a. de printer-status-poort;
   wanneer we gegevens willen afdrukken dan zal de waarde 'nul' op deze poort moeten worden geplaatst. Voor de gemiddelde MSX-computer is het poortnummer: &H90.

b. de printer-data-poort;

via deze poort worden gegevens getransporteerd. Voor de gemiddelde MSX-computer is het poortnummer &H91.

Met behulp van die kennis kunnen we een programma ontwerpen dat alleen gegevens uitvoert naar de printer als de waarde 'nul' op de printer-status-poort wordt 'gezet'.

Voorbeeld:

```
10 CLS
15 INPUT "PRINTEN J/N";Y$
20 IF Y$="J" OR Y$="j" THEN PR=0 ELSE PR=1
25 A$="TEST"
30 FOR I=1 TO LEN(A$)
35 KAR=ASC(MID$(A$,I,1))
40 GOSUB 50:NEXT I
45 END
50 OUT &H91,KAR
55 OUT &H90,PR
60 RETURN
```

In regel 20 wordt de variabele PR de waarde 'nul' gegeven als er gegevens geprint moeten worden; in het andere geval zal de variabele PR de waarde 1 bevatten. In de regels 30 en 35 worden de karakters van de string "TEST" omgezet in ASCII-kode-getallen, waarna in regel 40 de printer-routine wordt aangeroepen. Deze routine zet het kode-getal op de data-poort (regel 50) en plaatst daarna de waarde van PR op de status-poort (regel 55). Als PR gelijk is aan 'nul' zal de tekst "TEST" worden afgedrukt; in het andere geval gebeurt er niets.

Machinetaal-ekwivalent:

De Z80-instruktie OUT (zie hoofdstuk 2.19).

De Z80-instruktie-set kent twee mogelijkheden om gegevens door te geven aan een poort:

a. via register C: OUT (C),register;

b. direkt, in samenwerking met register A:

OUT (poortnummer),A.

Willen we de subroutine uit het Basic-voorbeeldprogramma omzetten in machinetaal dan zullen we de volgende stappen moeten zetten:

```
10 ' ORG 49500
20 ' LD A,(KAR)    ;plaats inhoud 'variabele' KAR in A
30 ' OUT (&H91),A
40 ' LD A,(PR)     ;plaats inhoud 'variabele' PR in A
50 ' OUT (&H90),A
60 ' RET           ;naar Basic
70 ' KAR NOP
80 ' PR NOP
90 ' EIND END
```

De waarden van 'KAR' en 'PR' zullen vanuit het Basic-programma aan de machinetaal-routine doorgegeven moeten worden met behulp van twee POKE-instrukties. Het is een wat omslachtige procedure, die in dit specifieke geval weinig praktisch nut heeft, maar het gaat er hier om dat we leren inzien dat Basic-programma's en machinetaal-programma's uitstekend met elkaar kunnen samenwerken.

De voorbeeld-routine kan vanuit het Basic-programma worden aangeroepen door het 'GOSUB 55'-kommando te vervangen door de instruktie: 'Z=USR0(0)'.
Voorwaarde is wel dat de routine in het geheugen is geladen en dat het startadres ervan is doorgegeven aan de computer met behulp van het 'DEFUSR'-kommando.

### *3.21 PEEK (adres)*

Basic-funktie, waarmee de inhoud van het RAM- en ROM-geheugen kan worden uitgelezen. Het adres van de geheugenlokatie die men wil bekijken (0 - 65535) moet tussen haakjes worden geplaatst.

Voorbeeld:

```
10 FOR I=32768 TO 32800
20 PRINT PEEK(I);
30 NEXT
```

Na RUNnen van dit programma staat de in kode omgezette programmaregel 10 op het scherm.

Machinetaal-ekwivalent:

Die Z80-load-instrukties, die de inhoud van een geheugenlokatie, waarvan het adres in een 16-bits-register (of registerpaar) staat, in een 8-bits-register plaatsen (zie ook hoofdstuk 2.3):

```
I     LD rr,adres    (rr = BC, DE, HL)
      LD A,(rr)

II    LD HL,adres
      LD r,(HL)     (r = A, B, C, D, E, H, L)

III   LD rr,adres    (rr = IX of IY)
      LD r,(rr+d)    (d = verplaatsingsgetal)
```

Hoewel alle 8-bits-registers gebruikt kunnen worden (in II en III) zal in de praktijk het A-register het meest gebruikt worden, omdat vrijwel alle rekenkundige en logische bewerkingen gebruik maken van dit register.

Voorbeeld:

Stel we willen in een Basic-programma een bepaald token (dat is een kode-getal dat verwijst naar een Basic-instruktie) opzoeken, bijvoorbeeld het token voor de instruktie: 'PRINT: 145'.

Een Basic-programma zou er als volgt uitzien:

```
10 FOR I=32768 TO 33000
20 IF PEEK(I)=145 THEN PRINT I:END
30 NEXT
```

Wanneer de geheugenlokatie, waarin zich de waarde 145 bevindt, gevonden is wordt het adres van de lokatie op het scherm gezet en

het programma stopt (regel 20).

Het assembler-bron-programma zou er zo uit kunnen zien:

```
10 ' ORG 49500
15 ' LD HL,32768      ;startadres in registerpaar HL
20 ' LD BC,33000      ;eindadres in registerpaar DE
25 ' LUS LD A,(HL)    ;A=PEEK(HL)
30 ' CP 145           ;vergelijk inhoud A met 145
35 ' JP NZ,NEXT       ;IF A<>145 THEN NEXT
40 ' LD (MEM),HL      ;IF A=HL THEN POKE MEM,HL
50 ' RET              ;return - terug naar Basic
55 ' NEXT INC HL      ;neem volgend adres HL+1
60 ' LD A,L           ;plaats inhoud register L in de akku
65 ' CP C             ;vergelijk met inhoud register C
70 ' JP NZ,LUS        ;IF HL<>BC THEN LUS
75 ' RET              ;IF HL=BC THEN RETURN
80 ' MEM DS 2         ;reserveer 2 geheugenplaatsen
85 ' EIND END
```

In de regels 60 en 65 worden alleen de registers L en C met elkaar vergeleken om te bepalen of HL na verhoging gelijk is aan BC. Dat is hier toegestaan omdat de inhouden van de registers B en C reeds gelijk zijn aan elkaar. We kunnen dit kontroleren door de getallen 32768 en 33000 om te zetten in hexadecimale vorm met behulp van de opdracht: PRINT HEX$(getal):

32768 is hexadecimaal: &H8000 (H=&H80 L=&H00)
33000 is hexadecimaal: &H80E8 (H=&H80 L=&HE8)

Wanneer het getal 145 wordt aangetroffen in het geheugen wordt de instruktie in regel 40 uitgevoerd. Het adres van de betreffende geheugenlokatie wordt in het door ons gereserveerde geheugengebied MEM geplaatst, dat kan worden uitgelezen met behulp van het volgende Basic-kommando:

```
PRINT PEEK(MEM)+256*PEEK(MEM+1)
```

We moeten hier kiezen voor een omweg, omdat het niet mogelijk is getallen die groter zijn dan 9 rechtstreeks op het beeldscherm af te drukken. We zullen daarvoor zelf een routine moeten ontwerpen,

hetgeen een tamelijk gekompliceerde zaak is, of we zullen onze toevlucht moeten nemen tot de print-routines, die in het Basic-ROM aanwezig zijn (zie daarvoor het volgende hoofdstuk, waarin een aantal Basic-ROM-routines worden besproken).

***Let op:***
Getallen van 0 tot en met 9 kunnen direkt op het scherm worden geplaatst door er de waarde 48 bij op te tellen. Er ontstaat in dat geval de ASCII-kode-getallenreeks 48 tot en met 57. Grotere getallen kunnen slechts worden afgedrukt door het getal via deel- en aftrek-procedures om te zetten in printbare eenheden, bijvoorbeeld:

```
354/100=3  354-3*100=54  54/10=5  54-5*10=4
```

### *3.22 POKE adres,in te voeren waarde*

Basic-kommando, waarmee bepaalde waarden direkt in het RAM-geheugen kunnen worden 'geschreven'. De waarde die men aan een geheugenlokatie wil geven wordt (gescheiden door een komma) achter het opgegeven adres geplaatst.

Voorbeeld:

```
POKE 50000,42
```

De waarde 42 wordt in lokatie 50000 geplaatst.

Het is niet mogelijk op een willekeurige wijze gegevens in het geheugen te POKEn, omdat men in dat geval het gevaar loopt in te breken in Basic-programma's of gedeelten van het geheugen die worden gebruikt door ROM-routines of (wanneer men gebruik maakt van het FLASH-assembler-pakket) het assembler-programma.

In het algemeen kan men stellen dat er altijd een vrij geheugengebied moet worden opgezocht of gekreëerd, wanneer men gegevens in het geheugen weg wil schrijven.

Het Basic-kommando 'CLEAR' biedt de mogelijkheid om zelf een dergelijk gebied in het leven te roepen. We moeten, wanneer we die

instruktie gebruiken, eerst weten wat het hoogste adres is in het RAM-geheugen dat door de Basic-interpreter wordt gebruikt. Dat adres treffen we aan in de systeemvariabele &HFC4A (HIMEM). De aangetroffen waarde zal niet gelijk zijn voor alle machines.

In het algemeen gelden de volgende waarden (voor machines die gebruik maken van een diskdrive):

MSX1: 57977
MSX2: 56953

Willen we een vrij geheugengebied kreëren van 1000 bytes dan zullen we de computer de volgende opdracht moeten geven:

MSX1: CLEAR 200,56977
MSX2: CLEAR 200,55953

Gegevens kunnen daarna probleemloos in dit vrije gebied worden geplaatst.

**Let op:**
De gebruikers van het FLASH-assembler/disassembler-pakket hebben een vrij geheugengebied tot hun beschikking dat zich uitstrekt vanaf het adres 49500 tot de HIMEM-waarde die bij hun machine behoort (zie hierboven). Dit gebied kan voor allerlei experimentele doeleinden worden gebruikt, onder andere voor de opslag van scherm-pagina's uit het video-RAM.

Machinetaal-ekwivalent:

De Z80-load-instrukties die de inhoud van een 8-bits-register in een geheugenlokatie plaatsen, waarvan het adres is toegekend aan een 16-bits-register of registerpaar (zie ook hoofdstuk 2.3).

```
I     LD rr,adres   (rr = BC, DE, HL)
      LD A,getal/inhoud geheugenlokatie
      LD (rr),A

II    LD HL,adres
      LD r,getal/inhoud geheugenlokatie
      LD (HL),r   (r = A, B, C, D, E, H, L)
```

III LD rr,adres (rr = IX of IY)
LD r,getal/inhoud geheugenlokatie
LD (rr+d),r (d = relatieve verplaatsing)

Stel we willen het PRINT-token (kodegetal: 145) in een Basic-programmaregel vervangen door een LPRINT-token (kodegetal: 157). Een Basic-programma zou er als volgt uit kunnen zien:

Voorbeeld:

```
10 PRINT "*"
20 ST=32768 'beginadres Basic-programma
30 FOR I=0 TO 20
40 IF PEEK(ST+I) <> 145 THEN 60
50 POKE ST+I,157:END
60 NEXT
```

Na het RUNnen van dit programma zal het PRINT-kommando in regel 10 zijn omgezet in een LPRINT-kommando. We hebben dit bewerkstelligd door de inhoud van slechts één geheugenlokatie te veranderen!

Het assembler-bronprogramma dat een token moet vervangen wordt volgens hetzelfde principe opgebouwd:

Voorbeeld:

```
10 ' ORG 49500
15 ' LD HL,32768        ;beginadres Basic in HL
20 ' LD B,20            ;gebruik register B als teller
25 ' LUS LD A,(HL)      ;A=PEEK(HL)
30 ' CP 145             ;vergelijk inhoud A met waarde 145
35 ' JP NZ,NEXT         ;IF A<>145 THEN NEXT
40 ' LD A,157           ;plaats LPRINT-token in de akku
45 ' LD (HL),A          ;POKE HL,A
50 ' RET                ;return - terug naar Basic
55 ' NEXT INC HL        ;neem volgende adres: HL+1
60 ' DJNZ LUS           ;B=B-1:IF B<>0 THEN LUS
65 ' RET                ;return - terug naar Basic
70 ' EIND END
```

Nadat we dit programma met behulp van de assembler een plaats in het geheugen hebben gegeven kunnen we het volgende Basic-aanroep-programma intoetsen:

```
1 PRINT "*"
2 DEFUSR0=49500
3 Z=USR0(0)
4 END
```

Wanneer we dit programma RUNnen zal blijken dat ook in dit geval het PRINT-kommando in regel 10 is omgezet in een LPRINT-kommando.

Let op: een wat omvangrijker test-programma, waarin het bovenstaande voorbeeld is verwerkt, is terug te vinden in hoofdstuk 6.11.

### *3.23 PRINT*

Basic-kommando, waarvan de belangrijkste funktie de uitvoer van tekst naar het scherm is. Omdat MSX-Basic verschillende scherm-modes kent (tekst-mode en grafische mode) zijn er ook verschillende print-routines:

a. in tekst-mode volstaat de simpele print-opdracht; voorbeeld: PRINT "TEST".

b. in grafische mode moet eerst een 'bestand' worden geopend, waarna de tekst als een file wordt uitgevoerd naar het grafische scherm; voorbeeld:

```
10 SCREEN 2
20 OPEN "GRP:" AS 1
30 PSET (50,50) 'vul scherm-koordinaten in
40 PRINT#1,"TEST"
50 GOTO 50 'houd grafisch scherm in stand
```

Machinetaal-ekwivalent:

Tekst-mode:

De routines voor uitvoer van gegevens naar scherm en printer kwamen al eerder ter sprake. Zie de besprekingen van de Basic-instrukties 'CHR$' en 'LPRINT'.

Hier willen we een standaard-recept geven voor het plaatsen van tekst op het scherm met bebulp van BIOS-ROM-routines:

a.1 plaats aan het eind van het bronprogramma, na de RET*urn*-opdracht, de af te drukken tekst, met daarvoor geplaatst een label en voeg een eindmarkerings-byte toe; voorbeeld:
```
50 ' TEKST DM "TEST"
55 ' NOP ;nul = eindmarkering
```

a.2 plaats binnen het programma het startadres van het 'tekst-gebied' in een registerpaar; voorbeeld:
```
15 ' LD HL,TEKST
```

a.3 plaats de inhoud van de geheugenlokatie waarnaar HL verwijst in de akku (register A) en geef met behulp van een label (adreswjzer) aan dat een programma-lus wordt ontworpen; voorbeeld:
```
20 ' LUS A,(HL)
```

a.4 roep de gewenste print-routine aan; voorbeeld:
```
25 ' RST &H18 ;routine OUTDO
```

a.5 verhoog HL met 1 (HL = HL + 1); voorbeeld:
```
30 ' INC HL
```

a.6 vergelijk de inhoud van A met de waarde van de eindmarkerings-hyte (hier 'nul'); voorbeeld:
```
35 ' CP 0 ; compare = vergelijk
```

a.7 herhaal de reeks opdrachten tot de inhoud van A gelijk is aan de opgegeven kontrole-waarde (hier 'nul'); voorbeeld:
```
40 ' JP NZ,LUS
```

Het volgende programma ontstaat op die manier:

```
10 ' ORG 49500
15 ' LD HL,TEKST
```

```
20 ' LUS LD A,(HL)
25 ' RST &H18
30 ' INC HL
35 ' CP 0
40 ' JP NZ,LUS
45 ' RET ;return
50 ' TEKST DM "TEST"
55 ' NOP
60 ' EIND END
```

Het gebruik van de standaard-routine OUTDO wordt aanbevolen, omdat met behulp van die routine tekst zowel naar het scherm als naar de printer kan worden uitgevoerd (zie 'CHR$').

Grafisch Scherm:

Het plaatsen van tekst op het grafische scherm kan tot stand worden gebracht met behulp van de BIOS-ROM-routine &H008D (GRPPRT).

De volgende stappen moeten worden ondernomen:

b.1 aktiveer (zonodig) het grafische scherm (zie 'SCREEN'); voorbeeld:
```
40 ' CALL &H0075 ; screen 3
```

b.2 bepaal de positie waar de tekst geplaatst moet worden. Dat kunnen we bewerkstelligen door de gewenste x- en y-koördinaten eerst te plaatsen in een registerpaar (bijvoorbeeld HL) en vervolgens de inhoud van dit registerpaar door te geven aan de volgende systeemvariabelen:
&HFCB7 (GRPACX) - 'kolom'-nummer
&HFCB9 (GRPACY) - 'regel'-nummer

b.3 geef de opdracht tot printen; voorbeeld:
```
70 ' CALL &H008D
```

b.4 konstrueer een lus die het grafische scherm in stand houdt, maar die ***niet*** eindeloos is; voorbeeld:
```
75 ' LUS CALL &H00BA ; roep STOP-routine &H00BA aan
80 ' JP LUS          ;spring naar begin van LUS
```

door het tegelijk indrukken van de toetsen Ctrl en Stop kan uit deze lus gesprongen worden!

Het volledige programma ziet er als volgt uit:

Voorbeeld:

```
10 ' ORG 49500
15 ' GRPACX EQU &HFCB7
20 ' GRPACY EQU &HFCB9
25 ' GRPPRT EQU &H008D
30 ' STOP EQU &H00BA
35 ' ;
40 ' START CALL &H0075      ; screen 3
45 ' LD HL,110             ; x-koordinaat
50 ' LD (GRPACX),HL
55 ' LD HL,80              ; y-koordinaat
60 ' LD (GRPACY),HL
65 ' LD A,42               ; ASCII-waarde van '*' in A
70 ' CALL GRPPRT           ; print het sterretje
75 ' LUS CALL STOP
80 ' JP LUS
85 ' EIND END
```

Deze routine zet het scherm in grafische mode 3 en plaatst een sterretje op het scherm op een positie die wordt aangegeven door de koordinaten x en y.
In de regels 75 en 80 wordt het grafische scherm in stand gehouden door het kontinu aanroepen van de BIOS-STOP-routine.

### *3.24 PSET (x,y),kleur*

Basic-kommando, waarmee een stip of punt op het grafische scherm geplaatst kan worden (SCREEN 2 en 3).

De waarde van x kan liggen tussen 0 en 255;
de waarde van y kan liggen tussen 0 en 192.

Voorbeeld:

```
10 SCREEN 3
20 PSET (163,47),1
30 GOTO 30
```

De waarde van x is 163, y heeft de waarde 47 en de kleurkode is 1, zodat op de door x en y aangegeven positie een zwarte stip wordt afgedrukt.

Machinetaal-ekwivalent:

de BIOS-ROM-routines:

&H011A (SETATR): bepaalt de kleur, de kleurkode (0-15) wordt in de akku verwacht.

H010E (SCALXY): zorgt ervoor dat de ingevoerde x- en y-waarden binnen het toegestane bereik vallen. (De routine wijzigt AF.)

&H0111 (MAPXYC): berekent de Punt-positie. (Wijzigt: AF,HL,D.)

&H0120 (SETC): zet de punt op het scherm. (Wijzigt: AF,EI.)

De x-koördinaat wordt in registerpaar BC verwacht; de y-koördinaat in registerpaar DE.

Voorbeeld:

```
10 ' ORG 49500
15 ' INIMLT EQU &H0075
20 ' STOP EQU &H00BA
25 ' SETATR EQU &H011A
30 ' SCALXY EQU &H010E
35 ' MAPXYC EQU &H0111
40 ' SETC EQU &H0120
45 ' ;
50 ' START CALL INIMLT  ;screen 3
55 ' LD A,1            ;kleurkode 1 (zwart) in de akku
60 ' CALL SETATR       ;vul de kleur in
65 ' LD BC,163         ;x-koordinaat
70 ' LD DE,47          ;y-koordinaat
```

```
75 ' CALL SCALXY          ;pas (zonodig) schaal aan
80 ' CALL MAPXYC          ;bereken de Punt-positie
85 ' CALL SETC            ;zet de Punt op het scherm
90 ' LUS CALL STOP        ;-> deze lus houdt het
95 ' JP LUS               ;grafische scherm in stand
99 ' EIND END
```

In regel 50 wordt de computer in scherm-mode 3 (de veelkleuren-mode) gezet. In de regels 55 en 60 wordt de kleur van de punt bepaald. In de regels 65 en 70 worden de x- en y-koördinaten ingevoerd. In de regels 75 tot en met 85 voert de computer de benodigde berekeningen uit die ertoe leiden dat de punt op het scherm wordt gezet.

Het grafische scherm wordt in stand gehouden door de programmalus in de regels 90 en 95, waarin de STOP-routine &H00BA wordt aangeroepen. De routine plaatst (net als het Basic-programma) een zwarte stip op het grafische scherm op de aangegeven positie.

Door de waarden in BC en DE te variëren kunnen stippen (of reeksen van stippen) op elke gewenste plaats worden afgebeeld.

Let op: hoofdstuk 6.12 bevat een wat uitvoeriger test-programma.

### *3.25 PUT SPRITE spritenummer,(x,y),kleur,patroonnummer*

Basic-kommando, dat de gebruiker in staat stelt een sprite op de diverse schermen (met uitzondering van het tekst-scherm) te plaatsen.

Het sprite-nummer, de schermpositie, de kleur en het nummer van het patroon in de 'patroon-tabel' moeten (in de hierboven weergegeven volgorde) aan het kommando worden toegevoegd.

Omdat we het Basic-kommando willen omzetten in een overeenkomstige machinetaal-routine zullen we ons moeten afvragen wat er nu precies gebeurt wanneer we het kommando 'PUT SPRITE' uitvoeren. We dienen ons dan te realiseren dat er in het video-RAM twee tabellen zijn (de extra mogelijkheden van de MSX2-computer laten we hier buiten beschouwing) die het werken

met sprites mogelijk maken:

a. de sprite-patroon-tabel op adres 14336:
b. de sprite-attribuut-tabel op adres 6912.

De attribuut-tabel is opgebouwd uit blokjes van 4 bytes, die de 'attributen' van een sprite bevatten, dus de scherm-koördinaten (x en y), de kleurkode en het patroonnummer.
De eerste 4 bytes (hokje 1) horen bij sprite 0; de tweede 4 bytes horen bij sprite 1; enzovoort (zie de figuur).

```
SPRITE 0                                 SPRITE 1
-----------------------------------------------------------------
 Y-koor     X-koor     patroon    kleur    Y    ... enzovoort.
 dinaat     dinaat     nummer     nummer
 0-192      0-255      0-255      0-15
-----------------------------------------------------------------
 6912       6913       6914       6915     6916
```

Wanneer het 'PUT SPRITE'-kommando wordt uitgevoerd dan zullen de geheugenlokaties in het video-RAM (die bij de gedefinieerde sprite behoren) met de ingevoerde 'attribuut-waarden' worden opgevuld. Dat invullen van geheugenlokaties kunnen we ook zelf doen met hehulp van het Basic-VPOKE-kommando.

Voorbeeld:

```
10 SCREEN 3:CLS
50 VPOKE 6912,100 'y-koordinaat
60 VPOKE 6913,125 'x-koordinaat
70 VPOKE 6914,0 ;patroon-nummer
80 VPOKE 6915,15 ;kleur-kode (wit)
90 END
```

Deze vier programmaregels vormen een 'vertaling' van het kommando:

```
PUT SPRITE 0,(125,100),15,0
```

Plaatsen we de waarde 208 in adres 6912 (de byte die de y-waarde bepaalt) dan zullen alle sprites onzichtbaar worden; plaatsen we de

waarde 208 in adres 6916 dan zullen alle sprites, met uitzondering van sprite 0 onzichtbaar worden; enzovoort.

De vorm van de sprite wordt vastgelegd in de patroontabel (zie de Basic-Instruktie 'SPRITE$'). De adressen 14336 tot 14344 horen bij sprite 0; de adressen 14344 tot 14352 horen bij sprite 1; enzovoort. Willen we de sprite een blokvorm geven dan vullen we de bytes van de patroontabel met de waarde 255. In ons voorbeeldprogramma doen we dat op de volgende manier:

```
20 FOR I=0 TO 7
30 VPOKE 14336+I,255
40 NEXT I
```

RUNnen we dit kleine test-programma dan zal een witte, blokvormige sprite op het scherm verschijnen.

Machinetaal-ekwivalent:

Een machinetaal-routine zal, zoals we dat hierboven hebben gedaan, de sprite-tabellen in het video-RAM direkt moeten benaderen. We gebruiken daarvoor de Z80-Load-instrukties, waarmee we waarden in registers kunnen plaatsen, de INC(rement)-instruktie, waarmee we naar een volgend adres kunnen springen en de standaard-routine &H004D (WRTVRM), waarmee we waarden (die in de akku zijn geplaatst) in het video-RAM kunnen schrijven.

Alles wat we hoeven te doen is het omzetten van de Basic-VPOKE-instrukties in overeenkomstige assembler-instrukties:

Voorbeeld:

```
10 ' ORG 49500
15 ' LD HL,6912    ;startadres attribuut-tabel
20 ' LD A,100      ;y-koordinaat in de akku
25 ' CALL &H004D   ;VPOKE 6912,100
30 ' INC HL        ;HL wordt 6913
35 ' LD A,125      ;x-koordinaat
40 ' CALL &H004D   ;VPOKE 6913,125
45 ' INC HL        ;HL wordt 6914
50 ' LD A,0        ;patroonnummer in de akku
```

```
55 ' CALL &H004D  ;VPOKE 6914,0
60 ' INC HL       ;HL wordt 6915
65 ' LD A,15      ;kleurkode in de akku
70 ' CALL &H004D  ;VPOKE 6915,15
75 ' RET          ;terug naar Basic
80 ' EIND END
```

Nadat we deze routine een plaats in het geheugen hebben gegeven met behulp van de assembler kunnen we het volgende aanroep-programma uitvoeren.

Aanroep:

```
10 SCREEN 1:DEFUSR0=49500:CLS
20 FOR I=0 TO 7
30 VPOKE 14336+I,255
40 NEXT
50 Z=USR0(0)
60 END
```

In regel 50 wordt de machinetaal-routine aangeroepen met behulp van de USR-funktie. De computer voert de machinetaal-routine uit en keert na uitvoering terug om verder te gaan met het Basic-programma. De machinetaal-routine fungeert dus als 'sub-routine' binnen het Basic-hoofdprogramma.

**Let op:**
Het MSX-BIOS-ROM bevat een standaardroutine die het adres van een attribuut-blok in het video-RAM kan berekenen: &H0087 (CALATR).

Het sprite-nummer wordt in register A (de akku) verwacht. Na uitvoering zal het adres van het blok in registerpaar HL staan.

De routine wijzigt: AF, DE, HL.

### *3.26 SCREEN scherminstelling*

Basic-kommando, waarmee (onder andere) de beeldscherminstelling kan worden gewijzigd.

Instellingen die zowel op MSX1 als MSX2-computers voorhanden zijn:

a. SCREEN 0, tekst-mode:
MSX1: 24 regels van 40 tekens;
MSX2: 24 regels van 40 - 80 tekens.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H006C (INITXT), aan te roepen met de Z80-instruktie CALL.

b. SCREEN 1, tekst + sprites:
24 regels van 32 tekens.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H006F (INIT32).

c. SCREEN 2, grafisch scherm:
opgebouwd uit 192 lijnen, die elk 32 bytes, dat is 32*8=256 bits, bevatten. bytes en bits kunnen afzonderlijk worden beïnvloed.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H0072 (INTGRP).

d. SCREEN 3, veelkleurenmode:
grafisch scherm dat is opgebouwd uit blokjes (of pixels), die 4 bits hoog en 4 bits breed zijn:
horizontaal: 64 pixels (64*4=256 bits)
vertikaal: 48 pixels (48*4=192 bits).

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H0075 (INIMLT).

Alle SCREEN-routines wijzigen de registers: AF, BC, DE, HL en schakelen de interruptverwerker in (EI).

### *3.27 SOUND registernummer,registerinhoud*

Basic-kommando, waarmee de registers van de geluidsgenerator (PSG = programmable sound generator) ingesteld kunnen worden, zodat één of meer geluidskanalen een door ons gewenst geluid gaan produceren.

| De registers van de PSG-geluidsgenerator | |
|---|---|
| 1. registers 0 - 5<br>bepalen de hoogte<br>van de toon<br>of de frekwentie | 0 en 1: kanaal A<br>2 en 3: kanaal B<br>4 en 5: kanaal C<br>in te voeren getallen:<br>0 - 4095 |
| 2. register 6<br>bepaalt toonhoogte<br>van ruis-geluiden | ruis-generator<br>in te voeren getallen:<br>0 - 31 |
| 3. register 7<br>'aan/uit'-register | bits 0-2 regelen toon A,B en C<br>bits 3-5 regelen ruis A,B en C<br>0=aan en 1=uit |
| 4. registers 8 - 10<br>bepalen het volume<br>of de geluidssterkte | 8=A, 9=B en 10=C<br>in te voeren getallen:<br>0 - 15 |
| 5. registers 11 en 12<br>bepalen lengte toon | in te voeren getallen: 1-65535<br>lage deel in 11; hoge in 12 |
| 6. register 13<br>bepaalt de golf-vorm<br>dus: soort geluid | <br>in te voeren getallen:<br>8 - 15 |

*figuur 3.27a*

```
--------------------------------------------------------------
Het AAN/UIT-register van de geluidsgenerator:
--------------------------------------------------------------
```

| bit7 | bit6 | bit5 | bit4 | bit3 | bit2 | bit1 | bit0 | |
|---|---|---|---|---|---|---|---|---|
| xxx | xxx | ruis C | ruis B | ruis A | toon C | toon B | toon A | kanaal |
| 1 | 0 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0/1 | 0=aan/1=uit |
| 128 | 064 | 032 | 016 | 008 | 004 | 002 | 001 | |

```
de bits 6 en 7 moeten de aangegeven waarde behouden
--------------------------------------------------------------
```

*figuur 3.27b*

Bij het programmeren van de geluidsgenerator dient men in het algemeen de volgende stappen te zetten:

a. het AAN/UIT-register, register 7, zodanig instellen dat het geluidskanaal dat men wenst te beïnvloeden 'aan' of 'uit' wordt gezet;

b. instellen van de toonhoogte, de geluidssterkte, de 'lengte' van de toon en de golfvorm;

c. het uitzetten (eventueel aanzetten) van het geluidskanaal dat men heeft 'geprogrammeerd' door beïnvloeding van het AAN/UIT-register (register 7).

Het PSG-register 7 kent zes schakelmogelijkheden, die betrekking hebben op de drie geluidskanalen waarover de generator beschikt (aan te duiden met de letters A, B en C). Willen we kanaal A een toon laten produceren dan moet bit 0 van register 7 laag (0) worden gemaakt. We zouden dit kunnen bewerkstelligen door in het schakelregister de waarde 'nul' te plaatsen, maar het resultaat daarvan zal zijn dat alle toon- en ruiskanalen worden ingeschakeld, hetgeen niet onze bedoeling was.

Een beter resultaat zullen we bereiken wanneer we een eenvoudig optelsommetje konstrueren:

```
Toon-kanaal A aan: bit 0 laag - decimale waarde: 000
Toon-kanaal B uit: bit 1 hoog - decimale waarde: 002
Toon-kanaal C uit: bit 2 hoog - decimale waarde: 004
Ruis-kanaal A uit: bit 3 hoog - decimale waarde: 008
Ruis-kanaal B uit: bit 4 hoog - decimale waarde: 016
Ruis-kanaal C uit: bit 5 hoog - decimale waarde: 032
Bit 6 moet altijd 0 bevatten  - decimale waarde: 000
Bit 7 moet altijd 1 bevatten  - decimale waarde: 128  +
-------------------------------------------------------
eindresultaat:                                   190
```

Het getal 190 zullen we in register 7 moeten 'laden'; hetgeen als volgt kan gebeuren: SOUND 7,190 Kanaal A zal nu (nadat de andere registers met de gewenste waarden zijn gevuld) een toon produceren.

Willen we toonkanaal A weer uitschakelen dan zal bit 0 van het schakelregister hoog (1) moeten worden gemaakt. Kijken we naar het optelsommetje hierboven dan zien we dat het hoog maken van bit 0 de decimale waarde 1 aan het totaal zal toevoegen. De waarde 191 zal daarom in het schakelregister moeten worden geladen met behulp van het kommando SOUND 7,191.

De overige registers worden niet op een 'meervoudige wijze' gebruikt, zodat hun beïnvloeding vrij eenvoudig is (zie figuur 3.27a).

Voorbeeld:

```
10 SOUND 7,190  'toonkanaal A aan
20 SOUND 0,255  '0 en 1 bepalen de toonhoogte
30 SOUND 1,0
40 SOUND 8,10   'volume of geluidssterkte
50 A$=INPUT$(1) 'wacht op indrukken toets
60 SOUND 7,191  'schakel toonkanaal A uit
```

Machinetaal-ekwivalent:

Om het programmeren van de PSG of geluidsgenerator mogelijk te maken bevat het MSX-ROM-BIOS een aantal standaardroutines:

aa. &H0090 (GICINI) - initialiseert de geluidsgenerator (De routine schakelt de interruptverwerker in.)

bb. &H0093 (WRTPSG) - schrijft gegevens naar de PSG-registers; deze routine komt overeen met het Basic-kommando SOUND. Het PSG-register-nummer wordt in de akku verwacht; de inhoud van het PSG-register moet in het Z80-register E worden geplaatst.
Voorbeeld:

```
SOUND 6,15 (komt overeen met:) 10 ' LD A,6
                               20 ' LD E,15
                               30 ' CALL &H0093
```

cc. &H0096 (RDPSG) - leest de inhoud van een PSG-register uit. Het registernummer wordt in de akku verwacht; het resultaat wordt in de akku geplaatst.

Willen we een geluidskanaal in- of uitschakelen, dan zal in het algemeen de volgende procedure gevolgd moeten worden:

1. Lees het schakel-register (register 7) uit.
   Dat kan als volgt gebeuren:
   ```
   LD A,7
   CALL &H0096 ;lees register
   ```
   Het resultaat van de uitlezing komt in de akku te staan.

2. Bewerk de inhoud van de akku op zo'n manier dat het gewenste kanaal aan- of uitgezet wordt. Dit kan men op twee manieren bewerkstelligen:

   *2a.* Door gebruik te maken van de Z80-'SET'- en RES*et*-instrukties:
   bijvoorbeeld SET 0,A (schakel toonkanaal A uit);
   RES 0,A (schakel toonkanaal A in).

   *2b.* Door een 'masker-bewerking' uit te voeren met behulp van de logische instrukties 'AND' en 'OR' (zie hoofdstuk 1.8). UIT-zetten bereikt men met een OR-masker; AAN-zetten wordt bereikt met een AND-masker.

Stel we willen toonkanaal A en ruiskanaal A uitzetten. We maken dan het volgende 'byte-plaatje':

```
7   6   5   4   3   2   1   0       OR - MASKER
--------------------------------
1   0   0   0   1   0   0   1       decimale waarde: 137
--------------------------------
128 064 032 016 008 004 002 001
```

Het kenmerk van de OR-bewerking is dat een bewerking met de waarde 'nul' geen enkel effekt heeft: (1 OR 0) blijft 1 en (0 OR 0) blijft 0.
De bits die we *niet* willen beïnvloeden moeten in het 'masker' dus laag zijn.
De bits die we *wel* willen beïnvloeden maken we hoog, immers: (1 OR 1) blijft 1 en (0 OR 1) wordt 1.
De inhoud van de akku zullen we nu met dit masker (logisch) moeten bewerken. Dat kan heel simpel gebeuren met de Z80-instruktie 'OR 9'.

Willen we daarentegen toonkanaal A en ruiskanaal A aanzetten, dan zullen we een AND-masker moeten konstrueren:

```
7   6   5   4   3   2   1   0       AND - MASKER
--------------------------------
1   0   1   1   0   1   1   0       decimale waarde: 182
--------------------------------
128 064 032 016 008 004 002 001
```

Willen we de bits 0 en 3 van de akku-inhoud laag maken dan voeren we de logische bewerking 'AND 182' uit. Alle overige bits blijven gelijk, immers: (0 AND 1) blijft 0 en (1 AND 1) blijft 1.

***Let op:***
Wanneer de werking van de AND-, OR- en XOR-instrukties nog niet geheel duidelijk is dan wordt aangeraden nogmaals enige tests uit te voeren met behulp van het Basic-test-programma in hoofdstuk 6.2.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD A,7       ;plaats registernummer in de akku
30 ' CALL &H0096 ;lees de inhoud van register 7
40 ' AND 182      ;voer de 'masker'-bewerking uit
50 ' LD E,A       ;plaats het resultaat in Z80-register E
60 ' LD A,7       ;plaats het registernummer in de akku
70 ' CALL &H0093 ;zet ruis- en toonkanaal A aan
enzovoort
```

Kanaal A zal nu zowel een toon- als een ruisgeluid produceren, nadat de waarden voor de toonhoogte en de geluidssterkte zijn ingevuld. Omdat we een 'masker'-bewerking hebben uitgevoerd zal de instelling van de overige kanalen ongewijzigd zijn.

3. Wanneer het AAN/UIT-register is ingesteld kunnen de overige registers van de geluidsgenerator worden ingesteld (eveneens met behulp van de standaardroutine &H0093 (WRTPSG).

4. Indien nodig wordt opnieuw het AAN/UIT-register uitgelezen en gewijzigd.

*Let op:*
Het is aan te bevelen alvorens men in 'machinetaal' gaat werken met de geluidsgenerator eerst een proefprogramma te schrijven in Basic, om daarna, wanneer men tevreden is over het bereikte resultaat, die Basic-routine om te zetten in een machinetaal-routine. In hoofdstuk 5 zal (bij wijze van voorbeeld) worden uiteengezet hoe zo'n omzetting in zijn werk gaat.

### *3.28 SPRITE$ (spritenummer)*

Basic-instruktie, waarmee de vorm (het sprite-patroon) van een sprite kan worden bepaald.

Voorbeeld:

```
10 COLOR 1,7,7
20 SCREEN 1:A$=""
30 FOR I=0 TO 7
40 A$=A$+CHR$(255):NEXT
```

```
50 SPRITE$(0)=A$
60 PUT SPRITE 0,(100,100),15,0
70 A$=INPUT$(1)
80 SCREEN 0:LIST
```

In de regels 30 en 40 wordt een blok-patroon opgebouwd: 8 bytes, waarvan alle bits hoog (1) zijn: 8 maal &B11111111. In regel 50 wordt dit blok-patroon via de instruktie SPRITE$(0) in de sprite-patroon-tabel (in het video-RAM) geplaatst. We kunnen dit kontroleren door met behulp van de Basic-funktie VPEEK het betreffende geheugengebied uit te lezen. Het startadres van de sprite-patroon-tabel (SCREEN 1, 2 en 3) is 14336, zodat de volgende opdracht moet worden ingetoetst:

```
FOR I = 0 TO 7:PRINT VPEEK(14336+I);:NEXT
```

We zullen zien dat het getal 255 acht maal op het scherm zal worden worden afgedrukt.

Het zal duidelijk zijn dat een sprite-patroon ook rechtstreeks in het video-RAM kan worden geschreven. In dat geval moeten we de regels 30 en 40 in het voorbeeld-programma vervangen door de volgende regel:

```
30 FOR I=0 TO 7:VPOKE 14336+I,255:NEXT
```

Op deze manier kunnen we de genele patroon-tabel vullen met gegevens. Wanneer er gekozen wordt voor een sprite-formaat van 8 bij 8 bits dan kunnen 256 reeksen van 8 bytes worden gedefinieerd: dat is in totaal 256*8=2048 bytes.

Wordt er gekozen voor een sprite-formaat van 16 bij 16 bits dan kunnen er 64 reeksen van 32 bytes worden gedefinieerd. De eerste 32 bytes vormen in dat geval patroon 0; de tweede 32 bytes vormen patroon 1; enzovoort... (raadpleeg voor een gedetailleerde uitleg een Basic-handboek).

Machinetaal-ekwivalent:

Binnen een machinetaal-routine zal men de patroon-gegevens van de sprite direkt in de patroon-tabel moeten plaatsen, zoals dat hier-

boven werd gerealiseerd met behulp van het Basic-kommando VPOKE. We gebruiken daarvoor de Z80-load-instrukties en een BIOS-ROM-routine die 'blok-verplaatsing' mogelijk maakt.

Het MSX-BIOS-ROM bevat twee blokverplaatsings-routines, die voor ons doel geschikt zijn:

a. standaardroutine &H005C (LDIRVM);
   hiermee kan een blok gegevens vanuit het RAM-geheugen naar het video-RAM worden getransporteerd (de routine wijzigt: AF, BC, DE, HL, EI). zie voor een bespreking 'COPY' en 'VPOKE'.

b. standaardroutine &H0056 (FILVRM);
   hiermee kan een geheugengebied in het video-RAM worden gevuld met een bepaalde waarde (wijzigt: AF, BC, EI).

Omdat we in ons voorbeeldprogramma een 'blokvormige' sprite hanteren kunnen we gebruik maken van de standaardroutine FILVRM; er hoeft immers maar een enkele waarde (het getal 255) in de patroontabel geplaatst te worden!

Voorbeeld:

```
10 ' ORG 49500
20 ' LD A,255           ;vorm-getal in de akku
30 ' LD HL,14336        ;startadres patroontabel in HL
40 ' LD BC,8            ;lengte van het blok in BC
50 ' CALL &H0056        ;schrijf de gegevens naar het VRAM
60 ' RET                ;return - naar Basic
70 ' EIND END
```

Plaatsen we dit programma in het geheugen met behulp van de assembler, dan kan de machinetaal-routine vanuit een Basic-programma worden aangeroepen met behulp van de USR-funktie.

Willen we een sprite-patroon dat uit meer waarden bestaat doorgeven aan de patroontabel in het video-RAM dan gebruiken we routine &H005C (LDIRVM). Als voorbeeld geven we een routine die de MSX-karakterset in de patroontabel plaatst.

We kiezen voor dit voorbeeld omdat het ons de mogelijkheid biedt letters en kombinaties van letters over het scherm te laten bewegen met behulp van het 'PUT SPRITE'-kommando (of de overeenkomstige machinetaal-routine). Het startadres van de karakterset in het MSX-ROM halen we uit de systeemvariabele &HF920 (CGPNT).

Voorbeeld:

```
10 ' LD HL,(&HF920)     ;startadres karakterset in HL
20 ' LD DE,14336        ;startadres patroontabel in DE
30 ' LD BC,2048         ;lengte van te verplaatsen blok
40 ' CALL &H005C        ;roep LDIRVM aan
50 ' RET                ;return - naar Basic
60 ' EIND END
```

Nadat we deze routine in het geheugen hebben geplaatst kunnen we met behulp van het volgende Basic-programma de tekst "MSX" over het scherm laten bewegen:

Boorbeeld:

```
10 DEFUSR0=49500:Z=USR0(0)
20 COLOR 1,7,7:SCREEN 1
30 PUT SPRITE 0,(100,80),1,77
40 PUT SPRITE 1,(108,80),1,83
50 PUT SPRITE 2,(116,80),1,88
60 FOR X%=0 TO 239
70 X1%=X%+8: X2%=X%+16
80 VPOKE 6913,X%:VPOKE 6917,X1%:VPOKE 6921,X2%
90 NEXT: GOTO 50
```

We zien dat in de regels 30 - 50 de patroonummers 77, 83 en 88 worden ingevoerd. Deze nummers zijn de ASCII-waarden van de letters 'M', 'S' en 'X' (we hebben immers de komplete karakterset overgezet!). In regel 80 plaatsen we de x-waarden van de sprites direkt in het video-RAM met behulp van het VPOKE-kommando (zie 'PUT SPRITE').

De programmalus gebruikt als teller de integer-variabele X%. We kiezen voor een integer-variabele omdat de snelheid in dat geval het

grootst zal zijn.

Het spreekt vanzelf dat het gebruik van een machinetaal-routine de bewegingssnelheid aanzienlijk zal vergroten. Die routine, een vertaling van de regels 60 - 90, volgt hieronder:

Voorbeeld:

```
10 ' ORG 49500
15 ' START LD A,0 ;register A bevat de x-waarde
20 ' LUS PUSH AF ;bewaar de inhoud van register A
25 ' LD HL,6913  ;adres x-lokatie sprite 0 in HL
30 ' CALL &H004D ;VPOKE 6913,A
35 ' ADD A,8     ;plaats het 2e teken naast het 1e teken
40 ' LD HL,6917  ;adres x-lokatie sprite 1 in HL
45 ' CALL &H004D ;VPOKE 6917,A
50 ' ADD A,8     ;plaats het 3e teken naast het 2e teken
55 ' LD HL,6921  ;adres x-lokatie sprite 2 in HL
60 ' CALL &H004D ;VPOKE 6921,A
65 ' CALL PAUZE  ;roep vertragingslus aan
70 ' CALL &H00BA ;roep STOP-routine aan
75 ' POP AF      ;herstel de inhoud van A
80 ' INC A       ;A=A+1 -> verplaats de sprites 1 bit
85 ' JP LUS      ;herhaal de routine
86 ' ;
87 ' PAUZE LD DE,500 ;dit is de vertragingslus
88 ' LUS1 DEC DE ;DE=DE-1
89 ' LD A,D
90 ' OR D
91 ' JP NZ,LUS1  ;IF (D OR E)<>0 THEN LUS1
92 ' RET         ;return - terug naar hoofdprogramma
95 ' EIND END
```

In regel 65 wordt een vertragingslus aangeroepen. Dit is noodzakelijk omdat de snelheid zonder deze lus te groot zou worden. De snelheid is afhankelijk van de waarde die in registerpaar DE wordt geplaatst (zie regel 87).

Vergroten we de inhoud van DE dan zal de tekst zich langzaam over het scherm bewegen; verkleinen we de inhoud van DE dan zal de snelheid dienovereenkomstig toenemen.

De routine kan als subroutine binnen het Basic-programma worden aangeroepen. De regels 60 - 90 van het Basic-programma moeten daartoe vervangen worden door de regel:

```
60 DEFUSR0=0:Z=USR0(0)
```

Door het tegelijk indrukken van de toetsen Ctrl en Stop kan het machinetaal-programma worden beëindigd.

***Let op:***
Het MSX-BIOS-ROM bevat een routine, waarmee het adres van een sprite-patroon kan worden berekend: &H0084 (CALPAT). Het patroonnummer wordt in de akku verwacht. Het resultaat, het adres van het patroon, wordt in registerpaar HL geplaatst. De routine wijzigt: AF, DE, HL.

### *3.29 STICK (x)*

Basic-funktie, waarmee de instelling van de joy-stick (indikatie-waarde x = 1 of 2) of van de kursor-toetsen (indikatie-waarde x = 0) kan worden opgevraagd. Er zijn acht instellingswaarden mogelijk, die elk verwijzen naar een bepaalde richting, bijvoorbeeld 1 = omhoog, 5 = omlaag, 3 = rechts en 7 = links.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H005D (GTSTCK).
De indikatie-waarde (0 = kursor, 1 = joystick1, 2 = joystick2) wordt in de akku (register A) verwacht, waarna de routine wordt aangeroepen met de Z80-instruktie 'CALL'. Het resultaat wordt in register A geplaatst.

Voorbeeld:

```
10 ' ORG 49500
20 ' LUS LD A,0      ;indikatie-waarde = 0, dus kursor
30 ' CALL &H005D     ;GeT STiCK
40 ' CP 3            ;< toets?
50 ' RET Z           ;ja, dan naar Basic
60 ' CP 7            ;> toets?
```

```
70 ' CALL Z,&H00C0 ;ja, dan BEEP
80 ' JP LUS        ;herhaal de routine
90 ' EIND END
```

***Let op:***
Een uitgebreid testprogramma is te vinden in hoofdstuk 6.13.

### *3.30 STOP*

Basic-kommando, waarmee het Basic-programma kan worden beëindigd: ofwel door het opnemen van de instruktie 'STOP' in een programmaregel, ofwel door het indrukken van de Ctrl- en Stop-toetsen tijdens de uitvoering van het programma.

Machinetaal-ekwivalent:

Twee BIOS-ROM-routines, die binnen een programma-lus moeten worden opgenomen, zodat na het indrukken van de toetsen Ctrl en Stop het programma kan worden stopgezet:

a. &H00BA (ISCNIC);
deze standaardroutine werkt niet als de interruptverwerker wordt uitgeschakeld. (Wijzigt: AF, EI.)

b. &H00B7 (BREAKX);
tast direkt het toetsenbord af om te kijken of de toetsen Ctrl en Stop werden ingedrukt; wanneer het test-resultaat positief is (de toetsen werden dus ingedrukt) zal de carry-vlag hoog worden gemaakt. De routine zal daarom gevolgd moeten worden door een instruktie die de carry-vlag test (bijvoorbeeld 'RET C'). De routine werkt ook als de interruptverwerker wordt uitgeschakeld. (Wijzigt: AF.)

### *3.31 STRIG (x)*

Basic-funktie, waarmee kan worden getest of de spatiebalk (indikatie-waarde x = 0) of de vuurknop van de joystick (indikatie-waarde x = 1/3 of 2/4) werd ingedrukt.
De funktie kan twee waarden opleveren als resultaat: 0 en -1. De

waarde 'nul' komt overeen met het antwoord 'nee'; de waarde -1 komt overeen met het antwoord 'ja'.

Voorbeeld:

```
10 A=STRIG(0)
20 IF A=-1 THEN PRINT "JA"
30 IF A=0 THEN PRINT "NEE"
40 GOTO 10
```

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H00D8 (GTTRIG).
De indikatie-waarde (0 = spatie-balk, 1/3 = vuurknop joystick1, 2/4 = vuurknop joystick2) wordt in register A verwacht.
Het resultaat, 0 = 'nee' of 255 (tweekomplement-notatie van het getal -1) = 'ja', wordt in register A geplaatst.

Voorbeeld:

```
10 ' ORG 49500
20 ' LUS LD A,0     ;indikatie-waarde = 0 -> spatiebalk
30 ' CALL &H00D8    ;GeT TRIGger
40 ' CP 255         ;spatiebalk ingedrukt?
50 ' CALL Z,&H00C0  ;ja, dan BEEP
60 ' CALL &H00BA    ;roep STOP-routine aan
70 ' JP LUS         ;herhaal de routine
80 ' EIND END
```

Een uitgebreid testprogramma is te vinden in hoofdstuk 6.13.

## *3.32 SWAP*

Basic-kommando, waarmee de inhoud van twee variabelen (numerieke variabelen en string-variabelen) kan worden omgewisseld.

Voorbeeld:

```
10 A=10:B=20:SWAP A,B
20 PRINT "A =";A;
```

```
30 PRINT "B =";B
```

Na RUNnen van het programma zal de tekst "A = 20 B = 10" op het scherm verschijnen.

Machinetaal-ekwivalent:

De Z80-instruktie EX*change* (zie hoofdstuk 2.5).

De belangrijkste exchange-instrukties zijn:

a. EXX - plaatst de inhouden van de registerparen BC, DE en HL in de hulp-registers BC', DE' en HL'.
   In Basic zouden we de volgende reeks instrukties moeten gebruiken: SWAP BC,BC': SWAP DE,DE': SWAP HL,HL'

b. EX DE,HL - (SWAP DE,HL) - verwisselt de inhouden van de registerparen HL en DE.

c. EX AF,AF' - (SWAP AF,AF') - verwisselt de inhouden van register A en het vlag-register (F) met de inhouden van de hulp-registers.

Omdat de Z80-processor geen string-variabelen kent zal het omwisselen van in het geheugen opgeslagen tekst-regels niet via een simpele SWAP- of exchange-procedure kunnen verlopen. Er zal in dat geval een verwissel-routine moeten worden geschreven, die in drie stappen moet verlopen:

1. verplaatsen tekst 1 naar een gereserveerd geheugengebied (buffer);

2. verplaatsen tekst 2 naar gebied 1;

3. verplaatsen tekst in buffer naar gebied 2.

In Basic zou zo'n verplaatsingsproces er als volgt uitzien:

Voorbeeld:

```
10 T1$="TEST1"
```

```
20 T2$="TEST2"
30 BUF$="T1$"
40 T1$=T2$
50 T2$=BUF$
```

Voor het verplaatsen van tekst kunnen in machinetaal-programma's de blokverplaatsingsroutines worden gebruikt (zie hoofdstuk 2.18).

### *3.33 VPEEK (adres)*

Basic-funktie, waarmee geheugenlokaties in het video-RAM kunnen worden bekeken.
Het adres (0 - 16384 voor de MSX1-computers en 0 - 65535 voor de MSX2-computers) moet tussen haakjes worden geplaatst.

Voorbeeld:

```
10 SCREEN 0:WIDTH 40
20 LOCATE 15,6:PRINT "*"
30 PRINT VPEEK(15+6*40+1)
```

RUNnen van dit programma zal de ASCII-waarde van bet sterretje, dat in regel 20 werd geprint, op het scherm zetten. De positie in het video-RAM wordt gevonden door het aantal regels (hier 6) te vermenigvuldigen met het aantal tekens dat de regel bevat (hier 40) en het aantal kolommen (hier 15) bij het resultaat op te tellen.
De waarde 1 moet er tenslotte aan toegevoegd worden om de juiste positie van het afgedrukte teken te vinden. Willen we het teken op het scherm zetten (in plaats van de ASCII-waarde) dan zullen we regel 30 de volgende vorm moeten geven:

```
30 PRINT CHR$(VPEEK(15+6*40+1))
```

De 'VPEEK-funktie' stelt ons in staat de inhoud van een scherm dat in een Basic-programma werd opgebouwd uit te lezen en in het RAM-geheugen te plaatsen met behulp van het 'VPOKE-kommando'.
Omdat dit uitlezen nogal wat tijd in beslag neemt is het weinig zinvol om binnen een Basic-programma op deze wijze gegevens te bewaren. Een machinetaal-routine kan hier uitkomst bieden.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H004A (RDVRM).
(Wijzigt: AF, EI.)

Het adres van het uit te lezen byte moet in het registerpaar HL worden geplaatst, waarna de routine kan worden aangeroepen met de Z80-instruktie 'CALL'.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD HL,256   ;15+6*40+1
30 ' CALL &H004A ;roep RDVRM aan
40 ' RST &H18    ;print het gevonden teken
50 ' RET         ;return - terug naar Basic
60 ' EIND END
```

Deze routine leest (wanneer het scherm in de 40-tekens tekstmode is gezet) het teken dat zich bevindt in regel 6, kolom 15 (in 80 tekensmode: regel 3, kolom 15). De ASCII-waarde van dat teken wordt in de akku (register A) geladen, zodat de print-routine (regel 40) het teken direkt op het scherm kan afdrukken.

Willen we meer tekens afdrukken dan zullen we de lees-routine RDVRM moeten opnemen in een programma-lus (zie hoofdstuk 3.9: 'FOR...NEXT').

***Let op:***
Om het snel uitlezen van blokken gegevens mogelijk te maken beschikt het MSX-BIOS over een aantal blok-verplaatsings-routines. Deze routines worden besproken in hoofdstuk 3.6 ('COPY').

Een uitgebreid testprogramma, waarin deze blokverplaatsingsroutines worden uitgetest, is te vinden in hoofdstuk 6.9

### *3.34 VPOKE adres*

Basic-kommando, waarmee gegevens direkt in het video-RAM kunnen worden geplaatst.

MSX1: adres 0 - 16384;
MSX2: adres 0 - 65535.

Niet alleen in tekst-mode kunnen we tekens op het scherm plaatsen, ook in de grafische mode is dat mogelijk. Omdat in de grafische mode (screen 2) de MSX-karakterset niet in het video-RAM aanwezig is zullen we het karakter dat we op het scherm willen plaatsen eerst uit de in het ROM aanwezige karakterset moeten halen. Het startadres van de karakterset in het ROM is te vinden in de systeemvariabele &HF920 (CGPNT).

Ieder karakter in deze karakterset is opgebouwd uit acht bytes, waarvan de bits op zo'n manier zijn gevuld met 'nullen' en 'enen' dat er een karakterpatroon ontstaat.

Voorbeeld:

```
karakter A        0 0 1 0 0 0 0 0    decimaal: 32
                  0 1 0 1 0 0 0 0              80
                  1 0 0 0 1 0 0 0              136
                  1 0 0 0 1 0 0 0              136
                  1 1 1 1 1 0 0 0              248
                  1 0 0 0 1 0 0 0              136
                  1 0 0 0 1 0 0 0              136
                  0 0 0 0 0 0 0 0              0
```

De acht decimale waarden (32 - 0) die bij het teken 'A' behoren vinden we in de karakterset door de ASCII-waarde van het teken (65) met acht te vermenigvuldigen. Het adres in het ROM vinden we door de gevonden waarde op te tellen bij het startadres van de karakterset.

Voorbeeld:

```
10 KARSET=PEEK(&HF920) + 256*PEEK(&HF921)
20 ST=KARSET + 65*8
30 FOR I=0 TO 7
40 PRINT PEEK(ST+I):NEXT
```

Als alles goed is verlopen zullen de hierboven afgebeelde decimale waarden op het scherm verschijnen.

Willen we het teken 'A' op het grafische scherm zetten, dan zullen we de uitgelezen decimale waarden met behulp van het VPOKE-kommando in de patroon-tabel van het video-RAM moeten plaatsen. De regels 30 en 40 in het voorbeeldprogramma moeten in dat geval worden vervangen door de volgende regels:

```
30 COLOR 1,7,7:SCREEN 2
40 FOR I=0 TO 7
50 VPOKE 6*256+15*8+I,PEEK(ST+I)
60 NEXT
70 GOTO 70
```

Het karakter A wordt op het grafische scherm (screen 2) gezet op een plaats die als volgt wordt berekend:

X-koördinaat = kolomnummer maal 8
Y-koördinaat = regelnummer maal 256.

Het grafische scherm kan in totaal 768 tekens bevatten (24 regels van 32 bytes). De patroon-tabel heeft daarom een lengte van 768 maal 8 = 6144 bytes.

Machinetaal-ekwivalent:

De BIOS-ROM-routine &H004D (WRTVRM).
(Wijzigt: AF, EI.)

Het startadres in het video-RAM moet in registerpaar HL worden geplaatst. Het gegeven dat in het video-RAM geschreven moet worden, wordt in de akku verwacht.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD HL,415     ; 10*40+15
30 ' LD A,65       ; ASCII-waarde van het teken 'A'
40 ' CALL &H004D   ; VPOKE 415,65
50 ' RET           ; return - naar Basic
60 ' EIND END
```

Op regel 10, in kolom 15, verschijnt (tekstmode: 40 tekens) het karakter 'A'.

Omdat we machinetaal vooral gebruiken vanwege de grotere snelheid waarmee gegevens kunnen worden gemanipuleerd, zullen we in de praktijk vooral gebruik maken van de blokverplaatsingsroutines waarover het MSX-ROM beschikt.

De routine die een blok gegevens naar het video-RAM schrijft is de standaardroutine &H005C (LDIRVM).

Het doeladres in het VRAM wordt in registerpaar DE verwacht; het adres van het te verplaatsen RAM-(of ROM)-geheugenblok moet in HL worden geladen, terwijl de lengte van het blok in registerpaar BC moet worden geplaatst.

Met behulp van deze blokverplaatsingsroutine kunnen we karakters vanuit het MSX-ROM naar het video-RAM schrijven (zoals we dat hierboven in een Basic-voorbeeldprogramma hebben gedaan).

Voorbeeld:

```
10 ' ORG 49500
20 ' LD HL,(&HF920) ;startadres karakterset in HL
30 ' LD BC,520      ;startadres 'A' in karakterset (65*8)
40 ' ADD HL,BC      ;HL=HL+BC -> startadres 'A' in ROM
50 ' LD BC,8        ;aantal bytes waaruit A is opgebouwd
60 ' LD DE,2680     ;10*256+15*8 -> scherm-positie
70 ' CALL &H005C    ;roep routine LDIRVM aan
80 ' RET            ;return - naar Basic
90 ' EIND END
```

In regel 20 wordt de inhoud van de systeemvariabele &HF920 uitgelezen om het startadres van de karakterset te bepalen. In de regels 30 en 40 wordt het adres berekend van het eerste byte dat bij het karakter 'A' behoort. (We zien dat eenvoudige optelsommen uitgevoerd kunnen worden met behulp van de 'ADD'-instruktie.) Lengte en schermpositie worden doorgegeven en tenslotte wordt de blokverplaatsingsroutine aangeroepen.

Deze machinetaal-routine kunnen we als volgt aanroepen binnen een Basic-programma:

```
10 DEFUSR0=49500
20 COLOR 1,7,7:SCREEN 2
30 Z=USR0(0)
40 GOTO 40
```

Door de waarde in *DE* te variëren kan het karakter op iedere willekeurige lokatie van het grafische scherm worden geplaatst.

***Let op:***
De machinetaal-routines die in dit hoofdstuk werden opgenomen kunnen de grondslag vormen van zelfgeschreven programma's. Er is dus duidelijk niet gestreefd naar volledigheid, omdat te lange programma's de fundamentele principes alleen maar zouden versluieren. Ook de wat langere test-programma's in het slothoofdstuk van dit boek zijn niet geschreven vanuit het oogpunt van 'praktisch nut'; ze willen in de eerste plaats de gebruiker ertoe aanzetten, zélf verder te experimenteren.
Het 'zelf-doen' maakt iemand tot programmeur; een goed leerboek kan en mag niet meer bieden dan een hoeveelheid 'basis-materiaal', waarmee een ieder op zijn eigen wijze verder kan werken.

# 4. Basic ROM-routines

## 4.1 Inleiding

In het vorige hoofdstuk hebben we kennisgemaakt met een groot aantal standaardroutines, die deel uitmaken van het MSX-BIOS-ROM. Deze routines vallen onder de MSX-standaard, hetgeen betekent dat ze gegarandeerd zullen werken op alle MSX1 en MSX2 computers. Zoals we weten bevat het MSX-ROM naast het BIOS gedeelte de Basic-interpreter: een geheel van machinetaalprogramma's, die in werking worden gesteld wanneer we een Basic-programma vervaardigen of RUNnen.

De routines uit dit Basic-ROM vallen niet onder de MSX-standaard. Dat betekent dat de aanroepadressen van deze routines in theorie voor iedere MSX-computer verschillend zouden kunnen zijn. In de praktijk echter zullen de verschillen tussen de diverse typen computers gering zijn. Het wijzigen van een Basic-vertaalprogramma is een uiterst moeilijke (en vooral kostbare) operatie, waar maar weinig fabrikanten brood in zullen zien. Het lijkt ons daarom zinvol een aantal van die Basic-routines uit het grote geheel te lichten, om op die manier het gebruik ervan in zelfgeschreven toepassingsprogramma's mogelijk te maken.

De beginnende programmeur stuit regelmatig op probleemsituaties, die hij onmogelijk kan oplossen met de kennis die hij op dat moment bezit. In dergelijke situaties is het uitermate handig terug te kunnen vallen op routines die al in de machine aanwezig zijn. Het proces van zelf programmeren wordt op die manier niet belemmerd, integendeel: de goede resultaten die men met behulp van de Basic-routines kan bereiken kunnen zelfs stimulerend werken en dat laatste is uitermate belangrijk.

We behandelen in de loop van dit hoofdstuk een aantal routines, die,

naar onze mening, het schrijven van toepassingsprogramma's zullen vergemakkelijken. Terwille van het gebruikersgemak hebben we getracht iedere routine een passende naam te geven:

ARGDAC - &H2705:
verplaatst buffer-inhoud

ASCDAC - &H3299:
zet ASCII-string om in een werkbaar getal

BASCP - &H2F83:
vergelijkt getallen in ARG en DAC

BASIC - &H411F:
initialiseert de Basic-interpreter

BASSRL - &H59B4:
verschuift de bits in registerpaar DE

CRLF - &H7328:
carriage-return + line-feed

DACARG - &H2F0D:
kopieert inhoud DAC in ARG

DACASC - &H3425:
zet een getal om in een ASCII-string

DACHEX - &H3722:
zet een getal om in een hexadecimale string

DACSGN - &H2EA1:
bepaalt het teken van een getal in DAC

DIVDAC - &H4DB8:
deelt twee getallen

ERROR - &H406F:
fout-verwerker

HLPRT - &H3412:
zet een getal in HL op het scherm

LINADR - &H4295:
zoekt het adres van een programmaregel

MLPDAC - &H3193:
vermenigvuldiging; resultaat in systeembuffer

MULTPL - &H314A:
vermenigvuldiging; resultaat in DE

STRPRT - &H6678:
print een string

VARADR - &H5EA4:
zoekt het adres van een variabele

VARINT - &H5439:
zet een getal in DAC om in een geheel getal

### *4.2 ARGDAC: aanroepadres &H2F05*

Deze routine kopieert de inhoud van de systeembuffer ARG (ook wel 'sekundaire akkumulator' genoemd) naar de systeembuffer DAC: een geheugengebied, dat door de Basic-interpreter bij alle berekeningen wordt gebruikt en dat daarom ook wel 'primaire akkumulator' wordt genoemd, om het verband aan te geven met het rekenregister van de Z80-processor: het A-register of de akkumulator.

Deze routine kan gebruikt worden (in kombinatie met routine DACARG) om na tijdelijke opslag van de inhoud van DAC in ARG de oorspronkelijke inhoud te herstellen.

***Toelichting:***

BUFFER: een gereserveerd geheugengebied waarin tijdelijk gegevens worden opgeslagen door de Basic-Interpreter en de BIOS-systeem-programma's.

De belangrijkste systeem-buffers zijn:

a. BUF: startadres &HF55E;
   in deze buffer wordt de tekst die van het toetsenbord wordt ingelezen opgeslagen.

b. KBUF: startadres &HF415;
   deze buffer bevat de gekodeerde versie van de ingetypte regel

c. DAC: startadres &HF7F6;
   hierin worden getallen (o.a. de inhoud van variabelen en adressen van variabelen) tijdelijk opgeslagen.
   Deze buffer kan een schakel vormen tussen machinetaalroutines en de USR-aanroep-funktie, omdat de parameter die (tussen haakjes geplaatst) aan deze funktie kan worden toegevoegd wordt doorgegeven aan de buffer, terwijl omgekeerd de inhoud van DAC via de USR-funktie kan worden doorgegeven aan een variabele (bijvoorbeeld Z = USR(0)).

d. ARG: startadres &HF847;
   deze buffer werkt samen met DAC, wanneer er berekeningen worden uitgevoerd door de Basic-interpreter.

e. FNKSTR: startadres &HF87F;
   deze buffer bevat de tekst van de funktie-toetsen (tien strings van zestien tekens).

### *4.3 ASCDAC: aanroepadres &H3299*

Wijzigt de inhoud van alle registers

Deze routine leest een in ASCII-tekens ingevoerd 'getal' en zet dit om in variabelen-notatie. De omzetting wordt beëindigd als er een 'niet-numeriek' teken in de string wordt aangetroffen.
Het startadres van de string wordt in registerpaar HL verwacht. Het resultaat wordt in de systeembuffer DAC geplaatst.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD HL,STRING       ;startadres string in HL
30 ' CALL &H3299        ;roep routine ASCDAC aan
40 ' RET                ;return - naar Basic-mode
50 ' STRING DM "25000"
60 ' NOP                ;eindmarkering
70 ' EIND END
```

Na vertaling van het bron-programma door de assembler en uitvoering van de routine met behulp van de opdracht:
'DEFUSR0=0:Z=USR0(0)' kan de juiste werking worden getest door de opdracht 'PRINT Z' in te toetsen. We zullen zien dat de variabele Z de waarde 25000 heeft gekregen: de inhoud van de systeembuffer DAC is doorgegeven aan het aanroep-programma!

***Let op:***
Een uitvoerig test-programma, waarin de routine ASCDAC gebruikt wordt om zelfingevoerde getallen bij elkaar op te tellen met behulp van Z80-rekeninstrukties is te vinden in hoofdstuk 6.15.

## *4.4 BASCP: aanroepadres &H2F83*

Deze routine vergelijkt twee getallen (die in dubbele precisie staan) met elkaar. Het eerste getal wordt in de systeembuffer DAC verwacht; het tweede getal in de buffer ARG.

Zijn de twee getallen gelijk dan wordt de zero-vlag hoog (1) gemaakt. Is het getal in DAC groter dan het getal in ARG dan wordt de carry-vlag hoog gemaakt; is het getal in DAC kleiner dan het getal in ARG dan wordt de carry-vlag laag (0) gemaakt.

## *4.5 BASIC: aanroepadres &H411F*

Het aanroepen van deze routine zal het initialiseren van de Basic-interpreter tot gevolg hebben. Wil men bijvoorbeeld vanuit een machinetaal-routine terugkeren naar de Basic-mode dan is dat vanuit iedere willekeurige plaats in het programma mogelijk met

behulp van de instrukties:

```
JP &H411F
```
*of*
```
CALL &H411F
```

***Let op:***
De MSX-computer biedt de gebruiker de mogelijkheid om de Basic-interpreter te initialiseren door het tegelijkertijd indrukken van de toetsen Ctrl, Graph, Code en Shift. Voorwaarde is dat de systeem-variabele &HFBB0 (ENSTOP) na het opstarten van de machine wordt gevuld met de waarde 1. Zelfs machinetaal-programma's kunnen op die manier in veel gevallen worden beëindigd wanneer het programma is vastgelopen!

### *4.6 BASSRL: aanroepadres &H59B4*

Deze routine verschuift alle bits in het registerpaar DE een plaats naar rechts (zie hoofdstuk 2.11: Roteer- en schuif-instrukties). Het praktische gevolg daarvan is dat de inhoud van registerpaar DE wordt gedeeld door twee. Bij het delen van oneven getallen wordt de carry-vlag hoog gemaakt.

Voorbeeld:

```
10 ' ORG 49500
15 ' VALTYP EQU &HF663
20 ' HLPRT EQU &H3412
25 ' BEEP EQU &H00C0
30 ' ;
35 ' START LD DE,64005
40 ' CALL &H59B4        ;roep routine BASSRL aan
50 ' PUSH DE            ;bewaar inhoud DE
55 ' CALL C,BEEP        ;was het getal oneven dan BEEP
60 ' POP HL             ;plaats inhoud DE in HL
65 ' CALL HLPRT         ;print het getal in HL
70 ' LD A,2             ;plaats integerkode 2 in de akku
75 ' LD (VALTYP),A      ;plaats de kode in VALTYP
80 ' RET                ;return - naar Basic
85 ' EIND END
```

De routine zal na vertaling en uitvoering het getal 32002 op het scherm zetten, terwijl tegelijkertijd een BEEP-toon wordt geproduceerd om aan te geven dat hier een oneven getal werd gedeeld (het resultaat van de deling is in dat geval niet korrekt; de waarde .5 zal aan het getal moeten worden toegevoegd).

Om te voorkomen dat de Basic-interpreter een foutmelding geeft wordt aan het eind van de routine de integerkode 2 in de systeem-variabele VALTYP geplaatst. Op die manier kunnen we veilig alle Basic-ROM-routines aanroepen met behulp van de opdracht 'Z=USR0(0)'.

De waarde van DAC wordt, zoals gezegd, toegekend aan de variabele die in de aanroep wordt gebruikt. Daarbij wordt de inhoud van VALTYP geraadpleegd. Bevat VALTYP de waarde 3 (de string-kode) dan zouden we, ter voorkoming van een foutmelding, een string-variabele in de aanroep moeten opnemen [b.v. Z$=USR0(0)]. Omdat dat in de praktijk nogal onhandig is hebben we voor deze oplossing gekozen.

### *4.7 CRLF: aanroepadres &H7328*

Wijzigt alleen AF.

Deze routine voert een 'carriage-return' en een 'linefeed'-opdracht uit; het praktische gevolg daarvan is dat naar een nieuwe regel wordt gesprongen.

### *4.8 DACARG: aanroepadres &H2F0D*

Kopieert de inhoud van de systeembuffer DAC (startadres &HF7F6) naar de systeembuffer ARG (startadres &HF847).

Deze routine kan worden toegepast in die situaties waarin we de inhoud van DAC willen bewaren (om bijvoorbeeld vergelijking mogelijk te maken; zie hoofdstuk 4.4).

### *4.9 DACASC: aanroepadres &H3425*

De routine wijzigt: AF, BC, DE, HL.

Een getal, dat zich in de systeembuffer DAC bevindt, wordt door deze routine omgezet in ASCII-tekens. De gevormde string wordt in de systeembuffer FBUFFR (startadres &HF7C5) geplaatst. Als eindmarkering wordt de waarde 'nul' aan de string toegvoegd. Het adres van de byte die voorafgaat aan de buffer (adres &HF7C4) staat na afloop in het registerpaar HL.

Voorbeeld:

```
10 ' ORG 49500
20 ' START CALL &H3425 ;zet het getal in DAC om
30 ' INC HL            ;maak HL gelijk aan &HF7C5
40 ' LUS LD A,(HL)     ;A = PEEK(HL)
50 ' CP 0              ;eindmarkering gevonden?
60 ' RET Z             ;ja, dan return - naar Basic
70 ' RST &H18          ;print inhoud register A
80 ' INC HL            ;neem volgend adres
90 ' JP LUS            ;herhaal de uitleesroutine
95 ' EIND END
```

Om het programma te testen ondernemen we de volgende stappen:

a. toets het kommando DEFUSR0=49500 in;

b. geef een getal (als parameter) door aan de machinetaalroutine met behulp van de funktie 'USR', bijvoorbeeld:
   `Z=USR0(35000).`

De machinetaalroutine zal nu het getal 35000 op het scherm printen, hetgeen bewijst dat het getal 35000 inderdaad werd doorgegeven aan de buffer DAC, waar het door de Basic-routine DACASC werd uitgelezen. De systeembuffer FBUFFR bevat de ASCII-waarden van de diverse getals-eenheden: 51 (cijfer 3), 53 (cijfer 5), enzovoort...

Het doorgeven van variabelen aan een machinetaal-routine via de systeembuffer DAC: startadres &HF7F6

1. numerieke variabelen:

| soort variabele | kode-nr. in &HF663 | plaats in de buffer |
|---|---|---|
| a. geheel of integer getal b.v. A% 0 tot +32767 -32768 tot 0 | 2 | &HF7F8 bevat lage byte &HF7F9 bevat hoge byte het getal kan direkt in in een Z80-registerpaar worden geladen |
| b. enkelvoudige precisie b.v. A! maximaal 6 cijfers | 4 | Lokaties &HF7F6 - &HF7F9 bevatten het getal in BCD-notatie het getal kan niet direkt worden verwerkt |
| c. dubbele precisie b.v. A# maximaal 14 cijfers | 8 | Lokaties &HF7F6 - &HF7FD bevatten het getal in BCD-notatie het getal kan niet direkt worden verwerkt |
| **2. string-variabelen** | | |
| b.v. A$ | 3 | Lokaties &HF7F8 en &HF7F9 bevatten het adres van de string-wijzer in het variabelen-geheugen |

**LET OP:**
Lokatie &HF7F6 bevat informatie over het teken en het aantal cijfers voor de 'komma' van de variabelen met enkelvoudige en dubbele precisie.
Is bit 7 hoog (1) dan is het getal negatief; is bit 7 laag (0) dan is het getal positief.

```
Het aantal cijfers voor de 'komma' wordt gevonden door
inhoud lokatie &HF7F6 te bewerken met het AND-masker 63.

b.v. LD A,(&HF7F6) ;inhoud lokatie in accu
     AND 63        ;A = (A AND 63)
     ADD A,48      ;zet om in ASCII-teken
     RST &H18      ;print het teken
     RET           ;return
-------------------------------------------------------
```

*figuur 4.9a*

## *4.10 DACHEX: aanroepadres &H3722*

Wijzigt: AF, BC, DE, HL.

Deze routine zet een getal dat zich in de systeembuffer DAC bevindt om in een 'hexadecimale string'. Het startadres van die string staat na afloop in het registerpaar HL. Als eindmarkering wordt het getal 'nul' aan de string toegevoegd.

Voorbeeld:

```
10 ' ORG 49500
20 ' START CALL &H3722 ;zet het getal in DAC om
30 ' CALL &H6678        ;printroutine STRPRT - zie 4.18
40 ' LD A,72          ;ASCII-waarde van 'H' in register A
50 ' RST &H18         ;print 'H'
60 ' LD A,2           ;2 = integerkode
70 ' LD (&HF663),A    ;in VALTYP
80 ' RET              ;return - naar Basic
90 ' EIND END
```

Met behulp van de funktie 'USR' kunnen we getallen doorgeven aan deze routine [bijvoorbeeld Z = USR0(35000)].
Als resultaat zal de hexadecimale waarde van het getal op het scherm worden geprint (regel 30), met als toevoeging de letter 'H' (regels 40 en 50).

***Let op:***
Een wat uitvoeriger test-programma is opgenomen in hoofdstuk 6.16.

### *4.11 DACSGN: aanroepadres &H2EA1*

Wijzigt: AF, BC, DE, HL.

Deze routine bepaalt het teken van het getal dat zich bevindt in de systeembuffer DAC. Het resultaat van het onderzoek wordt in de akku (register A) geplaatst, op de volgende manier:

| | |
|---|---|
| Positief: | inhoud A is 1 |
| Negatief: | inboud A is 255 (twee-komplement voor -1) |
| Getal is 'nul': | inhoud A is 'nul' |

Voorbeeld:

```
10 ' ORG 49500
15 ' START CALL &H2EA1 ;bepaal het teken
20 ' PUSH AF           ;bewaar het resultaat
25 ' CP 0              ;is het getal 'nul'?
30 ' CALL Z,&H00C0     ;ja, dan BEEP
35 ' POP AF            ;herstel inhoud AF
40 ' CP 1              ;getal positief?
45 ' RET Z             ;ja, dan terug naar Basic
50 ' CP 255            ;getal negatief?
55 ' CALL Z,&H00C3     ;ja, dan CLS
60 ' RET               ;return - naar Basic
65 ' EIND END
```

Met behulp van de USR-funktie kunnen we verschillende getallen doorgeven aan de routine. Is het getal dat we doorgeven 'nul' dan klinkt er een piep-toon. Is het getal positief dan gebeurt er niets. Is het getal negatief dan zal het scherm worden gewist.

### *4.12 DIVDAC: aanroepadres &H4D8B*

Wijzigt: AF, BC, DE, HL.

Twee gehele getallen, waarvan het eerste in registerpaar DE wordt verwacht en het tweede in registerpaar HL, worden gedeeld en het resultaat van de deling wordt in de systeembuffer DAC geplaatst (in enkele precisie).

Voorbeeld:

```
10 ' ORG 49500
20 ' START LD DE,1200  ;getal 1 in DE
30 ' LD HL,8           ;getal 2 in HL
40 ' CALL &H4DB8       ;deel de getallen
50 ' RET               ;return - naar Basic
60 ' EIND END
```

Na uitvoering van het programma met behulp van de instrukties
`DEFUSR0=49500`
en
`Z=USR0(0)`
zal de variabele Z de uitkomst van de deling bevatten: in dit geval het getal 150.

Een wat uitvoeriger test-programma is opgenomen in hoofdstuk 6.17.

### *4.13 ERROR: aanroepadres &H406F*

Op het adres &H406F start de fout-verwerkings-routine van de Basic-interpreter. Het nummer van de foutkode wordt in register E verwacht.

Mogelijke foutmeldingen en bijbehorende foutkode:

| | |
|---|---|
| Syntax Error: | 2 |
| Illegal Function Call: | 5 |
| Overflow: | 6 |
| Subscript out of range: | 9 |

| | |
|---|---|
| Out of memory: | 7 |
| Type Mismatch: | 13 |
| String too long: | 15 |
| Can't Continue: | 17 |

Voorbeeld:

```
10 ' ORG 49500
20 ' START LD E,17 ;Can't Continue-kode in E
30 ' JP &H406F     ;spring naar foutverwerker
40 ' EIND END
```

Uitvoering van dit programma zal ertoe leiden dat de tekst "Can't Continue" op het scherm wordt afgedrukt.

### *4.14 HLPRT: aanroepadres &H3412*

Wijzigt: AF, BC, DE, HL.

Met behulp van deze routine kan een getal dat zich in registerpaar HL bevindt op het scherm worden geprint.

Voorbeeld:

```
10 ' ORG 49500
20 ' START LD HL,12500
30 ' CALL &H3412       ;print inhoud HL
40 ' LD A,2            ;integerkode
50 ' LD (&HF663),A     ;in VALTYP
60 ' RET               ;return - naar Basic
70 ' EIND END
```

Uitvoering van het programma zal ertoe leiden dat het getal 12500 op het scherm verschijnt.

Met behulp van deze print-routine kan de inhoud van elk register worden afgedrukt. Voorwaarde is dat we de inhoud van het betreffende register in het registerpaar HL laden. Willen we bijvoorbeeld de inhoud van de akku printen, dan zullen we de volgende stappen moeten zetten:

Voorbeeld:

```
10 ' ORG 49500
20 ' LD A,115
30 ' LD H,0         ;hoge waarde wordt niet gebruikt
40 ' LD L,A         ;inhoud 8-bits-register A in L
50 ' CALL &H3412    ;print HL
60 ' LD A,2         ;integerkode
70 ' LD (&HF663),A  ;in VALTYP
80 ' RET            ;return
90 ' EIND END
```

De inhoud van een 8-bits-register moet altijd in register L (de lage of minst belangrijke byte) worden geplaatst. De waarde van register H moet in dat geval 'nul' worden gemaakt.

De inhoud van 16-bits-registerparen kan op de volgende manieren aan HL worden doorgegeven:

a. PUSH rr (rr = BC, DE, IX, IY)
   POP HL

b. LD H,hr (hoge deel register r in H)
   LD L,lr (lage deel register r in L)

c. EX DE,HL

## *4.15 LINADR: aanroepadres &H4295*

Wijzigt: AF, BC, DE, HL.

Deze routine zoekt het startadres van de programmaregel, waarvan het regelnummer in registerpaar DE is geplaatst. Wordt de regel gevonden, dan geeft de routine dat aan door het hoog (1) maken van de zero-vlag. Het startadres van de programmaregel wordt in dat geval in register paar BC geplaatst. Het adres van de programmaregel die er op volgt, wordt in registerpaar HL geplaatst. Dat betekent dat de lengte van de gezochte programmaregel gemakkelijk te achterhalen is: de waarde in BC moet afgetrokken worden van de waarde in HL.

Voorbeeld:

```
10 ' ORG 49500
20 ' LD DE,10          ;regelnummer 10 in DE
30 ' CALL &H4295       ;zoek de regel
40 ' RET NZ            ;zero-vlag laag? dan return
50 ' SBC HL,BC         ;bereken de regellengte
60 ' CALL &H3412       ;print de gevonden waarde
70 ' LD A,2            ;integerkode
80 ' LD (&HF663),A     ;in VALTYP
90 ' RET               ;return
95 ' EIND END
```

Dit programma zal, na uitvoering, de lengte van programmaregel 10 op het scherm plaatsen. Die lengte wordt berekend in regel 50 met behulp van de Z80-aftrek-instruktie 'SBC' (zie hoofdstuk 2.7).

Een wat uitvoeriger test-programma is opgenomen in hoofdstuk 6.18.

### *4.16 MLPDAC: aanroepadres &H3193*

Wijzigt: AF, BC, DE, HL.

De routine MLPDAC vermenigvuldigt de inhouden van de registerparen DE en HL met elkaar en plaatst het resultaat in de systeembuffer DAC:

- als geheel getal, wanneer het resultaat kleiner is dan 32768;

- in enkele precisie, wanneer het resultaat groter is dan 32767.

De getallen in DE en HL mogen niet groter zijn dan 32767: getallen die groter zijn dan 32767 worden namelijk als negatieve getallen behandeld, zodat het resultaat van de berekening 'onjuist' is.

Voorbeeld:

```
10 ' ORG 49500
20 ' START LD DE,1200
```

```
30 ' LD HL,5
40 ' CALL &H3193    ;vermenigvuldig
50 ' LD A,(&HF663) ;plaats inhoud VALTYP in akku
60 ' ADD A,48       ;zet getal om in ASCII-waarde
70 ' RST &H18       ;print inhoud A
80 ' RET            ;return
90 ' EIND END
```

Wanneer we dit programma uitvoeren met behulp van de opdracht

```
DEFUSR0=49500:Z=USR0(0)
```

zal het resultaat van de vermenigvuldiging via systeembuffer DAC worden doorgegeven aan de variabele Z, terwijl de inhoud van systeemvariabele VALTYP (die de variabelen-kode bevat) op het scherm wordt afgedrukt:

integerkode = 2, enkele precisie = 4, dubbele precisie = 8

In ons voorbeeld zal Z de waarde 6000 bevatten, terwijl op het scherm het getal 2 zal verschijnen.

Een iets uitvoeriger test-programma is opgenomen in hoofdstuk 6.19.

### *4.17 MULTPL: aanroepadres &H314A*

Wijzigt: AF, BC, DE, HL.

Deze routine vermenigvuldigt de gehele getallen die in de registerparen DE en BC staan en plaatst het resultaat van de berekening in registerpaar DE.

Is het resultaat groter dan 65535 dan wordt de foutmelding "Subscript out of range" gegeven.

Voorbeeld:

```
10 ' ORG 49500
20 ' START LD DE,20000
```

```
30 ' LD BC,5
40 ' CALL &H314A    ;vermenigvuldig BC en DE
50 ' EX DE,HL       ;SWAP DE,HL
60 ' CALL &H3412    ;print de inhoud van HL
70 ' LD A,2         ;integerkode
80 ' LD (&HF663),A  ;in VALTYP
90 ' RET            ;return
95 ' EIND END
```

Uitvoering van het programma zal een foutmelding opleveren, immers: het resultaat van de vermenigvuldiging (5 * 20000 = 100000) overschrijdt de toegestane grens van 65535. Geven we registerpaar DE een kleinere waarde, bijvoorbeeld 1200, dan zal het resultaat (6000) via de printroutine HLPRT gewoon op het scherm worden afgedrukt.

***Let op:***
In regel 50 wordt eerst de inhoud van DE in registerpaar HL geplaatst!

### *4.18 STRPRT: aanroepadres &H6678*

Wijzigt: AF, BC, DE, HL.

Met behulp van deze routine kan een string, waarvan het startadres in registerpaar HL moet worden geladen, op het beeldscherm worden afgedrukt. De string moet als eindmarkering het getal 'nul' of het getal 34 (ASCII-waarde van het aanhalingsteken ") bevatten.

Voorbeeld:

```
10 ' ORG 49500
20 ' START LD HL,TEKST ;adres string in HL
30 ' CALL &H6678       ;print de string
40 ' LD A,2            ;integerkode
50 ' LD (&HF663),A     ;in VALTYP
60 ' RET               ;return
70 ' TEKST DM "TEST"
80 ' DB 34             ;eindmarkering
90 ' EIND END
```

Uitvoering van het programma zal tot gevolg hebben dat de tekst "TEST" op het scherm verschijnt.

***Let op:***
In de regels 40 en 50 wordt de integerkode 2 in de systeemvariabele VALTYP geplaatst om een aanroep van het programma met behulp van een numerieke variabele mogelijk te maken. Omdat in dit programma een string wordt bewerkt zal VALTYP de waarde 3 bevatten. Dit betekent dat we de routine kunnen aanroepen met behulp van de volgende instruktie: Z$=USR0(0).
De regels 40 en 50 moeten bij het gebruik van deze aanroep uit het programma worden verwijderd! Voeren we het programma nu uit dan zal de tekst niet alleen worden afgedrukt maar ook toegekend aan de string-variabele Z$, hetgeen gekontroleerd kan worden door het kommando 'PRINT Z$' in te toetsen.

### *4.19 VARADR: aanroepadres &H5EA4*

Wijzigt alle registers.

Deze routine zoekt een variabele op in het variabelen-geheugen. Het startadres van het geheugengebied waarin zich de naam bevindt van de variabele moet in registerpaar HL worden geladen. Na uitvoering van de routine zal registerpaar DE het adres bevatten van het geheugengebied waarin zich de inhoud van de gezochte variabele bevindt. Registerpaar HL zal dan het adres bevatten van het byte dat volgt op het laatste teken van de naam van de variabele.

***Let op:***
Met behulp van deze routine kan de inhoud van een integer-variabele worden doorgegeven aan een Z80-registerpaar.

Voorbeeld:

```
10 ' ORG 49500
15 ' START LD HL,NAAM ;adres naam variabele in HL
20 ' CALL &H5EA4      ;zoek de variabele op
25 ' LD A,(DE)        ;plaats inhoud lage byte in A
30 ' LD L,A           ;plaats inhoud A in register L
35 ' INC DE           ;volgende byte
```

```
40 ' LD A,(DE)          ;plaats inhoud hoge byte in A
45 ' LD H,A             ;plaats inhoud A in register H
50 ' CALL &H3412        ;printroutine HLPRT
55 ' LD A,2             ;integerkode
60 ' LD (&HF663),A      ;in VALTYP
65 ' RET                ;return - naar Basic
70 '                    ;
75 ' NAAM DM "A%"       ;integervariabele
80 ' NOP                ;eindmarkering
85 ' EIND END
```

In regel 15 wordt het adres van het geheugengebied, waarin zich de naam van de integer-variabele bevindt (in dit geval A%), doorgegeven aan registerpaar HL.

Regel 75 bevat de naam van de variabele (die naam moet altijd uit hoofdletters bestaan; wanneer dat niet het geval is zal de computer een foutmelding geven!).

In regel 20 wordt de zoekroutine aangeroepen; deze routine zoekt het gebied in het geheugen waarin zich de inhoud van de variabele bevindt op en plaatst het adres van het lage byte in registerpaar DE.

In de regels 25 tot en met 45 wordt de inhoud van de integer-variabele in registerpaar HL geplaatst.
Die inhoud wordt tenslotte (ter kontrole) met behulp van de Basic-print-routine HLPRT op het scherm afgedrukt.

Voordat we nu dit programma gaan uitvoeren moeten we er eerst voor zorgen dat de integer-variabele A% wordt gedefinieerd (gevuld met een bepaalde inhoud).

Voorbeeld:

```
A%=125:DEFUSR0=49500:Z=USR0(0)
```

plaatst als resultaat het getal 125 op het beeldscherm, hetgeen betekent dat de inhoud van variabele A% inderdaad werd doorgegeven aan de machinetaalroutine.

Een uitvoerig test-programma is opgenomen in hoofdstuk 6.20.

### *4.20 VARINT: aanroepadres &H5439*

Wijzigt: AF, BC, DE, HL.

Deze routine zet een variabele, die zich in de systeembuffer DAC bevindt, om in een geheel getal en plaatst dat getal in registerpaar HL. De routine kan gebruikt worden in kombinatie met de routine ASCDAC (&H3299), die een ASCII-getallen-string omzet in variabelen-notatie.

Voorbeeld:

```
10 ' ORG 49500
20 ' CALL &H5439      ;omzetting -> geheel getal in HL
30 ' CALL &H3412      ;printroutine HLPRT
40 ' LD A,2           ;integerkode
50 ' LD (&HF663),A    ;in VALTYP
60 ' RET              ;return
70 ' EIND END
```

De juiste werking van de routine kan getest worden door met behulp van de 'USR-funktie' diverse getallen in de systeembuffer DAC te plaatsen.

Eerst definiëren we het funktie-adres: DEFUSR0=49500, daarna voeren we de getallen in:

voorbeeld:

a. integer-variabele: Z=USR0(1200%)
b. enkele precisie: Z=USR0(10000!)
c. dubbele precisie: Z=USR0(20000#)

Alle getallen zullen door de routine op het scherm worden geplaatst, hetgeen bewijst dat in alle gevallen een omzetting in gehele getallen heeft plaatsgevonden.

***Let op:***
Een uitvoerig test-programma, waarin de routine VARINT wordt gebruikt in kombinatie met de routine ASCDAC, is opgenomen in hoofdstuk 6.15.

# 5. Het vertalen van een Basic-programma

## *5.1 Inleiding*

In de voorafgaande hoofdstukken hebben we getracht een brug te slaan tussen MSX Basic en machine-(of assembler)-taal. We hebben aangetoond dat de stap van Basic naar machinetaal niet zo groot is als vaak wordt aangenomen.

Wanneer men zich eenmaal de grondbeginselen heeft eigen gemaakt is het vervaardigen van een machinetaal-programma geen onmogelijkheid wanneer men maar gebruik maakt van de handige ROM-routines die de MSX computer bevat.

In de meeste handboeken wordt alleen maar verwezen naar die routines, zonder ze te verbinden met het begrippen-apparaat dat de Basic-programmeur hanteert. Met dit boek hopen we de leemte die er op dit gebied bestond te hebben opgevuld, zodat machinetaal binnen het bereik van grotere aantallen computer-gebruikers kan komen.

Om de cirkel helemaal rond te maken willen we in dit hoofdstuk de kennis die de lezer in de vorige hoofdstukken heeft opgedaan gebruiken om komplete Basic-programmaatjes om te zetten in overeenkomstige machinetaal-routines.

Voor de Basic-programmeur die geïnteresseerd is in machinetaal lijkt ons dit de meest toepasselijke weg. Er zijn auteurs die als tussenstap het konstrueren van allerlei ingewikkelde diagrammen en schema's aanbevelen, maar wij zijn de mening toegedaan dat een eenvoudig, helder opgezet Basic-programma heel goed als tussenstap dienst kan doen.
Het gaat er om passende hulpmiddelen te vinden, die het programmeren tot een boeiende aangelegenheid kunnen maken. Een ieder

heeft het recht die hulpmiddelen te gebruiken die hem het meest aanspreken. Dat wordt in programmeurskringen nogal eens vergeten!

## *5.2 Het programmeren van de geluids-generator.*

In hoofdstuk 3 hebben we het Basic-kommando SOUND, waarmee de geluids-processor van de MSX-computer kan worden geprogrammeerd, besproken (zie hoofdstuk 3.27).
De overeenkomstige machinetaalroutines kwamen ter sprake en er werd verwezen naar de registers van de geluidsprocessor (zie figuur 3.27a en 3.27b).

We zullen hier een klein Basic-programma, dat het geluid van een huilende sirene produceert, volledig omzetten in een assembler-bron-programma:

### A. het Basic-programma

In het hiernavolgende Basic-programma wordt gebruik gemaakt van het 'kontrole-som-programma' uit hoofdstuk 6. De kontrole-getallen bevinden zich tussen haakjes en moeten ***niet*** worden ingetypt! Deze getallen kunnen tijdens het intypen van het programma worden vergeleken met de waarden die het 'kontrole-som-programma' op het scherm afdrukt, op de plaats waar zich gewoonlijk de tekst van funktietoets 2 bevindt.
Wanneer men de regel korrekt heeft ingetypt zal het getal dat op het beeldscherm wordt afgedrukt gelijk moeten zijn aan het kontrole-getal dat zich tussen haakjes in de programma-regel bevindt. Het spreekt vanzelf dat men eerst het 'kontrole-som-programma' in de computer moet laden, voordat men tot het intypen van dit programma overgaat (zie daarvoor hoofdstuk 6.1).

```
100 REM ==============================
110 REM   SOUND (SIRENE)
120 REM
130 REM   Basic-Programma
140 REM ==============================
150 REM
160 (51)  DEFINT I-J
```

```
170 (121) CLS:PRINT"SIRENE"
180 (40)  SOUND 7,VAL("&B10111011")
190 REM   BIT 2 laag: geluid C aan
200 REM   BIT 5 hoog: ruis C uit
210 (120) SOUND 5,0:SOUND 10,10
220 REM   toonhoogte C:registers 4 en 5
230 REM   geluidssterkte C:register 10
240 REM   LUS1->
250 (209) FOR I=1 TO 25
260 REM   LUS2->
270 (157) FOR J=255 TO 0 STEP -5
280 (82)  SOUND 4,J
290 (206) NEXT J
300 (205) NEXT I
```

We zullen dit Basic-programma stapsgewijs bespreken en de diverse onderdelen omzetten in overeenkomstige routines, die tesamen het bron-programma zullen vormen dat de assembler kan omzetten in machine-kode.

In regel 160 worden de variabelen I en J omgezet in integer-variabelen, die alleen gehele getallen kunnen verwerken. Een overeenkomstige machinetaal-routine geven we hier niet voor omdat de Z80-processor normaal gesproken alleen maar werkt met gehele getallen (behalve als we routines schrijven waarbij de BCD-kode wordt gebruikt; zie hoofdstuk 1.10 en 2.20).

In regel 170 wordt een wis-opdracht gegeven waarna de tekst "SIRENE" op het scherm wordt geprint. Met deze opdrachten zal ons machinetaal-programma moeten beginnen:

Voordat we echter de opdrachten aan de computer verstrekken gaan we eerst over tot het definiëren van labels met behulp van de assembler-pseudo-instruktie 'EQU'.

---

***Let op:***
De *pseudo*-instrukties worden in dit boek niet besproken, omdat er vanuit wordt gegaan dat de gebruiker beschikt over een assembler. met een bijbehorende toelichting. Het programma-pakket FLASH

(assembler en disassembler) bijvoorbeeld bevat een uitgebreide handleiding waarin de werking van deze instrukties aan de hand van test-voorbeelden wordt uitgelegd. Omdat wij in dit boek gebruik maken van deze assembler (die naar onze mening een ideaal instrument voor beginnende programmeurs is) willen we terwille van diegenen, die over een andere assembler beschikken, in het kort een opsomming geven van de pseudo-instrukties, die de FLASH-assembler gebruikt:

a. ORG (origine) b. EQU (equate) c. DM (define message)
d. DB (define byte) e. DW (define word) f. DS (define store)
g. END - afsluiting van het programma.

---

Het is een goede gewoonte om standaardroutines en (systeem)variabelen waarnaar in het bron-programma wordt verwezen in een label-lijst aan het begin van het programma op te nemen. Wanneer we het Basic-programma bekijken dan zien we dat we routines nodig hebben die:

a. het scherm wissen (standaardroutine CLS);

b. gegevens kunnen laden in de registers van de geluids-processor (standaardroutine WRTPSG) en

c. een tekst op het scherm kunnen afdrukken (Basic-routine STRPRT en de daarbij behorende variabele VALTYP).

Er ontstaat het volgende begin:

bronprogramma:

```
160 ' ORG 49500
170 ' VALTYP EQU &HF663
180 ' WRTPSG EQU &H0093
190 ' CLS EQU &H00C3
200 ' STRPRT EQU &H6678
210 ' ;
```

Regel 220 zal de wis-opdracht moeten bevatten. Daarna volgt de print-opdracht:

```
220 ' CALL CLS       ;wis het scherm
230 ' LD HL,TEKST
240 ' CALL STRPRT    ;print de tekst
250 ' ;
680 ' LD A,2         ;integerkode
690 ' LD (VALTYP),A
700 ' RET            ;return
710 ' ;
720 ' TEKST DM "SIRENE"
730 ' NOP
740 ' EINDE END
```

De af te drukken tekst wordt aan het einde van het programma geplaatst, voorafgegaan door een label, waarnaar in regel 230 wordt verwezen. De assembler plaatst het juiste adres in HL, hetgeen betekent dat voor ons de zaak rond is.

We hebben een programma geschreven dat het scherm wist en de tekst "SIRENE" afdrukt. Het programma heeft derhalve een kop en een staart, maar geluiden worden nog niet geproduceerd. Om dat te bereiken keren we terug naar het Basic-programma.

In de regels 180 tot en met 200 wordt duidelijk gemaakt dat het schakel-register van de geluidsprocessor (AAN/UIT-register 7; zie figuur 3.27b) op een zodanige wijze bewerkt moet worden dat kanaal C wel geluid, maar geen ruis gaat produceren.

Bit 2 van register 7 moet laag worden gemaakt (aan = 'nul').
Bit 5 van register 7 moet hoog worden gemaakt (uit = 1).

Terwille van de duidelijkheid wordt in regel 180 gekozen voor een binaire afbeelding van het getal, een konstruktie, die in ons assembler-programma natuurlijk niet gebruikt kan worden.
We kiezen hiervoor een procedure die is opgebouwd uit de volgende stappen:

a1. Ga er van uit dat alle ruis- en geluidskanalen moeten worden uitgeschakeld: dat betekent dat de laatste 6 bits (bit5 - bit0)

hoog moeten worden gemaakt, zodat de decimale waarde die naar het register geschreven moet worden gelijk wordt aan 128 (bit7 moet *altijd* hoog zijn; bit6 *altijd* laag) + 63 (32+16+8+4+2+1) = 191.

b1. Plaats deze waarde in de akku.

c1. Maak het bit dat het geluid van kanaal C regelt (bit2) laag met behulp van de Z80-RESet-instruktie: 'RES 2,A'.

d1. Schrijf de nieuwe waarde in PSG-register 7.

De volgende regels kunnen (wanneer we deze procedure volgen) aan het bronprogramma worden toegevoegd (zie ook hoofdstuk 3.27):

bronprogramma:

```
300 ' LD A,191      ;&H10111111
310 ' RES 2,A       ;maak bit2 laag ('nul')
320 ' LD E,A        ;plaats inhoud akku in register E
330 ' LD A,7        ;plaats nummer PSG-schakelregister in A
340 ' CALL WRTPSG ;schrijf inhoud E naar PSG-register
350 ' ;
```

Het Basic-programma geeft in regel 210 de opdracht de PSG-registers 5 (frekwentie - hoge byte) en 10 (volume) te vullen met gegevens. De omzetting in assembler-instrukties is hier eenvoudig:

bronprogramma:

```
360 ' LD E,0        ;plaats de waarde 'nul' in E
370 ' LD A,5        ;nummer PSG-register in de akku
380 ' CALL WRTPSG ;schrijf inhoud E naar PSG-register
390 ' LD E,10       ;volume = 10
400 ' LD A,10       ;nummer PSG-register in de akku
410 ' CALL WRTPSG ;schrijf naar PSG
420 ' ;
```

Moeilijker wordt de vertaling van de regels 240 - 300, waarin een SOUND-kommando in het hart van twee programma-lussen is

geplaatst. We beginnen met de eerste lus, die te vinden is in de regels 250 en 300:

```
'FOR I=1 TO 25' - 'NEXT I'.
```

Het betreft hier een eenvoudige telopdracht: tel van 1 tot 25 en stop als de teller de waarde 25 heeft bereikt.
We kiezen als machinetaal-ekwivalent de 'DJNZ'-konstruktie (zie hoofdstuk 3.9):

bronprogramma:

```
430 ' LD B,25    ;teller voor LUS1
440 ' LUS1
620 ' ;
630 ' DJNZ LUS1 ;B=B-1
640 ' ;              IF B <> 0 THEN LUS1
```

De machinetaal-lus vermindert voortdurend de waarde in register B en stopt als de inhoud van register B gelijk is aan nul.

Binnen deze eerste lus moet nu de tweede lus worden opgenomen:

`FOR J=255 TO 0 STEP -5` (regel 270) - `NEXT J` (regel 290).

Binnen de tweede lus wordt de teller regelmatig verlaagd met de waarde 5 en er wordt gestopt als de waarde van de teller gelijk is aan nul. We kiezen voor een machinetaal-konstruktie, waarbinnen register C als teller fungeert. Omdat de Z80-reken-instrukties via de akku verlopen zal de inhoud van register C tijdelijk overgebracht moeten worden naar de akku, waar de inhoud met de waarde 5 verminderd kan worden:

bronprogramma:

```
440 ' LUS1 LD C,255 ;beginwaarde 255 in C
450 ' ;
460 ' LUS2
570 ' ;
580 ' LD A,C        ;plaats inhoud register C in A
590 ' SUB 5         ;trek de waarde 5 er van af
```

```
600 ' LD C,A        ;plaats de nieuwe waarde in C
610 ' JP NC,LUS2    ;IF C>=0 THEN LUS2
```

In regel 610 van het bron-programma wordt gekeken of de carry-vlag laag is. Wanneer dat het geval is zal de routine, die start in regel 460, opnieuw worden uitgevoerd. De carry-vlag zal laag blijven zolang de inhoud van register A (regel 580) groter is dan 5.
Is de inhoud van A kleiner dan zal een situatie van '1 lenen' ontstaan (zie hoofdstuk 2.7) hetgeen het hoog (1) maken van de carry-vlag tot gevolg heeft. In dat geval zal de instruktie in regel 630 worden uitgevoerd.

De opdracht die binnen de twee lussen uitgevoerd moet worden is in het Basic-programma te vinden in regel 280:

```
SOUND 4,J
```

De toonhoogte van het geluid wordt door dit register bepaald (PSG-register 5 is gevuld met de waarde 'nul' en heeft derhalve geen invloed op het geproduceerde geluid). Omdat de waarde van variabele J varieert (afneemt van hoog naar laag) zal er een huiltoon ontstaan die langzaam uitsterft.

Om dat geleidelijke 'uitsterven' van de toon binnen het machinetaal-programma mogelijk te maken zullen we een vertragingslus moeten inbouwen in het programma. Zo'n vertragingslus is in feite een telopdracht. We laten de computer bijvoorbeeld van 1 tot 1000 tellen, zodat er een korte wachtpauze ontstaat.

We kunnen nu de volgende opdrachten toevoegen aan ons programma:

bronprogramma:

```
460 ' LUS2 LD E,C  ;inhoud teller C in register E
470 ' LD A,4       ;nummer PSG-register in de akku
480 ' CALL WRTPSG  ;schrijf naar PSG
490 '              ;
500 '              ;vertragingslus ->
510 ' LD HL,450    ;HL fungeert als teller
520 ' LUS3 DEC HL  ;HL=HL-1
```

```
530 ' LD A,H        ;voer een OR-bewerking uit
540 ' OR L
550 ' JP NZ,LUS3    ;IF (H OR L)<>0 THEN LUS3
```

De regels 460 tot en met 480 voeren de SOUND-opdracht uit. De vertragingslus zorgt ervoor dat het geluid niet onmiddellijk wordt afgebroken (door de waarde in HL te variëren kunnen we dus de tijdsduur bepalen).

Daarmee is het machinetaal-programma in feite voltooid. Om het opnemen van de routine als subroutine binnen een spel-programma mogelijk te maken is het zinvol om de inhoud van alle Z80-registers te 'bewaren' en aan het eind van het programma de oorspronkelijke inhoud te herstellen. Dat kan men bewerkstelligen met 'PUSH' en 'POP'-instrukties of met de 'EXX'-(exchange)-instruktie.

## B. het assembler-bron-programma

Het volledige programma ziet er als volgt uit:

```
100 ' ;==================================
110 ' ;Programmable SOUND Generator
120 ' ;
130 ' ;SIRENE: machinetaal-programma
140 ' ;==================================
150 ' ;
160 ' ORG 49500
170 ' VALTYP EQU &HF663
180 ' WRTPSG EQU &H0093
190 ' CLS EQU &H00C3
200 ' STRPRT EQU &H6678
210 ' ;
220 ' INIT SUB A    ;maak Zero-vlag hoog
230 ' CALL CLS      ;wis scherm
240 ' LD HL,TEKST
250 ' CALL STRPRT   ;Basic-printroutine
260 ' ;
270 ' START PUSH AF ;bewaar AF
280 ' EXX           ;bewaar registers BC,DE,HL
290 ' ;
300 ' LD A,191      ;&B10111111->A,B,C uit
```

```
310 ' RES 2,A        ;zet kanaal C
320 ' LD E,A         ;(geluid) aan
330 ' LD A,7         ;AAN/UIT-register = 7
340 ' CALL WRTPSG    ;schrijf naar PSG
350 ' ;
360 ' LD E,0
370 ' LD A,5         ;PSG-register 5
380 ' CALL WRTPSG    ;schrijf naar PSG
390 ' LD E,10        ;volume = 10
400 ' LD A,10        ;PSG-register 10
410 ' CALL WRTPSG    ;schrijf naar PSG
420 ' ;
430 ' LD B,25        ;teller voor lus1
440 ' LUS1 LD C,255
450 ' ;
460 ' LUS2 LD A,4    ;register 4
470 ' LD E,C         ;toonhoogte
480 ' CALL WRTPSG    ;schrijf naar PSG
490 ' ;
500 ' ;               vertragingslus ->
510 ' LD HL,450      ;teller = 450
520 ' LUS3 DEC HL    ;HL = HL-1
530 ' LD A,H         ;voer OR bewerking
540 ' OR L           ;uit
550 ' JP NZ,LUS3     ;IF (H OR L) <> 0 (NZ)
560 ' ;               THEN LUS3
570 ' ;
580 ' LD A,C         ;plaats inhoud C in akku
590 ' SUB 5          ;verminder met 5
600 ' LD C,A         ;plaats inhoud A in C
610 ' JP NC,LUS2     ;IF C >= 0 THEN LUS2
620 ' ;
630 ' DJNZ LUS1      ;B = B-1
640 ' ;               IF B <> 0 THEN LUS1
650 ' ;
660 ' EXX            ;herstel BC,DE,HL
670 ' POP AF         ;herstel AF
680 ' LD A,2         ;integerkode
690 ' LD (VALTYP),A
700 ' RET            ;RETURN
710 ' ;
```

```
720 ' TEKST DM "SIRENE"
730 ' NOP
740 ' EINDE END
750 ' ;
760 ' ;               stel routine in werking ->
770 DEFUSR0=49500!:Z=USR0(0)
780 END
```

In regel 770 is de 'USR'-aanroep-opdracht opgenomen, zodat het programma (nadat het vertaaid is) met behulp van het Basic-'RUN'-kommando in werking kan worden gesteld.

## *5.3 SCREEN 2: kleuren veranderen*

Wanneer we werken met het grafische scherm (SCREEN 2) van de MSX-computer moeten we rekening houden met drie tabellen in het video-RAM, die in onderlinge samenwerking het 'beeld' dat op het scherm verschijnt opbouwen:

a. de patroon-tabel (adres 0 - 6144);
b. de kleuren-tabel (adres 8192 - 14336);
c. de scherm-tabel (adres 6144 - 6912).

De scherm-tabel bepaalt welke tekens uit de patroon-tabel er op het beeld-scherm zullen verschijnen. Elke byte in deze tabel verwijst naar een groepje van 8 bytes in de patroontabel.

De kleurwaarden van de tekens op het scherm worden gereguleerd door de kleuren-tabel (zie figuur 5.3a).

De kleurentabel start op het adres 8192. De byte op dat adres bepaalt de kleur van de eerste acht beeldschermpunten. De tabel eindigt op het adres 14336. De byte op dat adres bepaalt de kleur van de laatste acht beeldschermpunten.

Het hoge deel van de byte (bit7 - bit4) bepaalt de voorgrondkleur. Wanneer we een kleurkode in dit gedeelte willen plaatsen zullen we het kode-getal eerst met 16 moeten vermenigvuldigen. Het lage deel van de byte (bit3 - bit0) bepaalt de achtergrondkleur.
We kunnen dit kontroleren door met behulp van het kommando

```
SCREEN 2
kleurentabel, begin-adres in VRAM: 8192
lengte: 6144

schermindeling:

0            <- y-as ->               31
---------------------------------------
8192  8200  8208  ...               8440   regel 1:
8193  8201  8209  ...               8441
8194  ...                           8442   32 blokken
f195  ...                           8443   van 8 bytes
8196  ...                           8444   of
8197  ...                           8445   256 bytes
8198  ...                           8446
8199  8207  8215  ...               8447
---------------------------------------
8448  8456  8464  ...               8796  regel 2
|     ...                           |
                                          startadres:
|     ...                           |     8192+1*256=8448
8455  8463  8471  ...               8703
---------------------------------------
|
regels 3 - 7
|
---------------------------------------
 984  9992  10000 ...              10232  regel 8
9985  9993  10001 ...              10233
9986  ...                          10234  startadres:
9987  ...                          10235  8192+7*256=9984
9988  ...                          10236
9989  ...                          10237
9990  ...                          10238
9991  ...                          10239
---------------------------------------
10240                              10488  regel 9
```
*figuur 5.3a*

'VPOKE' rechtstreeks waarden in de kleurentabel te schrijven.

Voorbeeld:

```
10 COLOR 5,5,5:SCREEN 2
20 FOR I=0 TO 15
30 VPOKE 8192+I,(16*1+1)     'zwart
40 VPOKE 14320+I,(16*15+15) 'wit
50 FOR J=1 TO 100:NEXT
60 NEXT I
70 GOTO 70
```

Wanneer we dit programma RUNnen, zullen we kunnen konstateren dat in de linker-bovenhoek en in de rechter-onderhoek van het beeldscherm stapsgewijs reeksen van acht beeldpunten worden ingekleurd. Bekijken we nu figuur 5.3a, dan zal ons onmiddellijk duidelijk worden, op welke wijze de MSX-computer bij het inkleuren van het scherm te werk gaat. Zoals we op het scherm reeds zagen, worden blokken van bytes (byte voor byte) ingekleurd. Wanneer een blok afgewerkt is, wordt er naar het begin van het volgende blok gesprongen. Wanneer er 32 blokken van 8 bytes zijn afgewerkt wordt naar een volgende regel gesprongen. Regelgewijs wordt het scherm dus ingekleurd wanneer we de opeenvolgende bytes van de kleurentabel vullen met een bepaalde waarde.

Teneinde de byte-opbouw van het grafische scherm duidelijk te maken en tegelijkertijd het verschil in snelheid tussen een Basic-invul-routine en een machinetaal-routine te laten zien, willen we in dit gedeelte van het boek een Basic-routine, die gedeelten van het scherm van kleur verandert, omzetten in een reeks assembler-instrukties.

### A. het Basic-programma

```
100 REM ================================
110 REM   KLEURENTABEL IN VRAM VULLEN
120 REM   met een bepaalde kleurwaarde
130 REM
140 REM   Basic-programma
150 REM ================================
160 (160) CLS
```

```
170 (120) COLOR1,14,14:SCREEN 2
180 (65)  OPEN"grp:" AS 1
190 (173) PRESET(25,50)
200 (149) PRINT#1,"MSX-BASIC EN ";
210 (87)  PRINT#1,"MACHINETAAL"
220 (11)  PRESET(60,110)
230 (82)  PRINT#1,"TEST-PROGRAMMA"
240 (19)  PRESET(60,118)
250 (41)  PRINT#1,"ctrl+stop=einde"
260 REM
270 (167) LE=2048 'LENGTE BLOK
280 REM   vul kleurentabel in VRAM
290 REM   met de waarde 245->
300 REM   wit/blauw:15(*16)+5
310 (193) BEEP
320 (56)  FOR I=0 TO LE
330 (112) VPOKE 8192+I,245
340 (132) NEXT
350 (111) FOR I=0 TO 250:NEXT 'PAUZE
360 REM   vul kleurentabel in VRAM
370 REM   met de waarde 17->
380 REM   zwart/zwart:1(*16)+1
390 (193) BEEP
400 (56)  FOR I=0 TO LE
410 (140) VPOKE 8192+I,17
420 (132) NEXT
430 (111) FOR I=0 TO 250:NEXT 'PAUZE
440 (168) GOTO 270
```

Ook dit programma bevat (geplaatst tussen haakjes). kontrolegetallen, zodat in samenwerking met het 'kontrole-som-programma' uit hoofdstuk 6.1 kan worden gekontroleerd of een regel juist is ingetypt. (Wanneer men kontrole niet nodig vindt hoeft met deze getallen verder geen rekening worden gehouden!)

Het eerste gedeelte van het programma zet het scherm in de grafische mode en plaatst een aantal tekst-regels op het scherm. Deze regels worden niet omgezet, omdat we er vanuit gaan dat de vertaling van deze regels (aan de hand van de informatie in hoofdstuk 3) weinig problemen zal opleveren.
In het 'aanroep-programma' dat is toegevoegd aan de machinetaal-

routine worden deze regels dan ook overgenomen (regels 690 - 780).

Het hoofd-programma, dat de eerste 2048 bytes van de kleurentabel met de gewenste kleurwaarden vult (de eerste 8 regels van het beeldscherm; zie fig. 5.3a), wordt begrensd door de programmaregels 270 en 440.

In regel 270 wordt de lengte van het te bewerken blok toegekend aan de variabele LE, waarna (nadat in regel 310 een BEEP-opdracht is gegeven) in de regels 320 - 340 de kleurentabel in het video-RAM met de kleurwaarde 245 wordt gevuld, met behulp van het Basic-kommando VPOKE.

We kunnen konkluderen dat de volgende waarden binnen het machinetaal-programma aan 'variabelen' toegekend zullen moeten worden: de lengte (2048), het beginadres van de tabel (8192) en de kleurwaarde (245).
Vervolgens moet de tabel worden gevuld met de gewenste kleurwaarde, waarvoor we de BIOS-ROM-routine FILVRM gebruiken, een routine die een blok geheugen in het video-RAM met een bepaalde waarde kan opvullen. De omzetting voert tot de volgende reeks assembler-instrukties:

bronprogramma:

```
230 ' START CALL STOP ;roep STOP-routine aan
240 ' CALL BEEP       ;roep BEEP-routine aan
270 ' LD A,245        ;plaats kleurwaarde in de akku
290 ' LD HL,8192      ;beginadres kleurentabel in HL
300 ' LD BC,2048      ;lengte van het blok in BC
310 ' CALL FILVRM     ;roep de vul-routine aan
```

Deze machinetaal-routine zal de eerste acht schermregels vullen met

---

*FILVRM, standaardroutine, aanroepadres: &H0056*
vult een blok gegevens in het video-RAM met een bepaalde waarde: lengte wordt in BC verwacht, doeladres in DE en weg te schrijven waarde in register A (de akku). Wijzigt: AF, BC, EI.

de kleuren blauw/wit.
De voorgrondkleur is wit, hetgeen betekent dat de letters van de tekst die op het scherm werd gezet een witte kleur zullen krijgen.

Kijken we naar het Basic-programma, dan zien we dat er na het inkleuren een wachtperiode moet worden ingelast. Dat gebeurt in programma-regel 350, waarin van 1 tot 250 wordt geteld.

Ook in regel 430 wordt een wachtperiode ingelast, die met behulp van dezelfde tel-opdracht wordt gerealiseerd. Omdat een pauze-lus in een assembler-programma nogal wat instrukties verlangt, zullen we deze lus als subroutine opnemen in het programma. Dat spaart ruimte en we voorkomen ermee dat we gelijke routines voortdurend moeten herschrijven.

Wanneer we de wachtlus als subroutine in het programma opnemen, kunnen we daar waar het nodig is de routine met behulp van de 80-instruktie ALL' aanroepen.

We voegen de volgende regels aan ons assembler-programma toe:

bronprogramma:

```
330 ' CALL REMLUS        ;roep de pauze-routine aan
450 ' JP START           ;herhaal de routine
460 ' ;
480 ' REMLUS PUSH BC
490 ' PUSH HL
500 ' LD B,3             ;teller 1
510 ' LUS1 LD HL,&HFFFF ;teller 2
520 ' ;
530 ' LUS2 DEC HL        ;HL = HL-1
540 ' LD A,H
550 ' OR L
560 ' JR NZ,LUS2         ;IF (H OR L)<>0 THEN LUS2
570 ' ;
580 ' DJNZ LUS1          ;B = B-1 (herhaal tot B=0)
590 ' POP HL
600 ' POP BC
610 ' RET                ;return: terug naar hoofdprogr.
```

Omdat in deze subroutine register B en registerpaar HL als teller worden gebruikt plaatsen we aan het begin van de routine de inhoud van deze registers met behulp van de stapel-instruktie 'PUSH' in het geheugen, om aan het einde van de routine de inhoud weer te herstellen met behulp van de instruktie POP.

Het is een goede gewoonte om de registerinhouden veilig te stellen, wanneer men een subroutine ontwerpt die regelmatig in een programma wordt aangeroepen. Op die manier kan men voorkomen dat het hoofdprogramma in de war raakt na een CALL-opdracht.

De pauze-lus bestaat uit twee lussen: LUS1 (roept driemaal LUS2 aan) en LUS2 (waarbinnen van 1 tot 65535 wordt geteld).
We zien onmiddellijk het verschil met de Basic-pauze-lus: in Basic tellen we van 1 tot 250; de machinetaal-routine telt 3 maal van 1 tot 65535 om ongeveer dezelfde wachttijd in het leven te kunnen roepen. De subroutine wordt aangeroepen in regel 330.

Na deze wachtperiode gaan we opnieuw de kleurentabel invullen met een kleurwaarde. Ditmaal maken we zowel de voorgrond als de achtergrond zwart, zodat de tekst van het scherm 'verdwijnt' (letters en achtergrond hebben dezelfde kleur; de tekst is dus niet echt verdwenen!).

In het Basic-programma treffen we deze opdracht aan in de regels 390 - 420. De overeenkomstige machinetaal-routine ziet er als volgt uit:

bronprogramma:

```
380 ' LD A,17      ;kleurwaarde in A
400 ' LD HL,8192   ;beginadres kleurentabel in HL
410 ' LD BC,2048   ;lengte van het blok in BC
420 ' CALL FILVRM ;roep de vul-routine aan
```

Omdat er opnieuw een wachtperiode moet worden ingelast (zie Basic-programmaregel 430) roepen we in het bron-programma de REMLUS aan:

```
440 ' CALL REMLUS
```

Daarmee hebben we de regels 270 - 440 van het Basic-programma vertaald in een machinetaal-programma (niet direkt, maar indirekt door het ontwerpen van een bron-programma voor de assembler).

## B. het assembler-bron-programma

```
100 ' ;==================================
110 ' ;  KLEURENTABEL IN VRAM OPVULLEN
120 ' ;  met een bepaalde kleurwaarde
130 ' ;
140 ' ;  machinetaal-programma
150 ' ;==================================
160 ' ;
170 ' ORG 49500
180 ' ;
190 ' BEEP EQU &H00C0
200 ' STOP EQU &H00BA
210 ' FILVRM EQU &H0056
220 ' ;
230 ' START CALL STOP
240 ' CALL BEEP
250 ' ;                     vul kleurentabel in VRAM
260 ' ;                     met de waarde 245->
270 ' LD A,245             ;kleuren
280 '                      ;wit/blauw:15(*16)+5
290 ' LD HL,8192           ;doel-adres in RAM
300 ' LD BC,2048           ;lengte van het blok
310 ' CALL FILVRM          ;vul-routine
320 ' ;
330 ' CALL REMLUS          ;pauze
340 ' CALL STOP
350 ' ;
360 ' ;                     vul kleurentabel in VRAM
370 ' ;                     met de waarde 17->
380 ' LD A,17              ;kleuren
390 ' ;                     zwart/zwart:1(*16)+1
400 ' LD HL,8192           ;doel-adres in VRAM
410 ' LD BC,2048           ;lengte van het blok
420 ' CALL FILVRM          ;vul-routine
430 ' ;
440 ' CALL REMLUS          ;pauze
```

```
450 ' JP START            ;herhaal
460 ' ;
470 ' ;                    vertragingslus(sen)->
480 ' REMLUS PUSH BC
490 ' PUSH HL
500 ' LD B,3              ;teller
510 ' LUS1 LD HL,&HFFFF
520 ' ;
530 ' LUS2 DEC HL         ;HL = HL-1
540 ' LD A,H
550 ' OR L                ;IF (H OR L) <> 0
560 ' JR NZ,LUS2          ;THEN LUS2
570 ' ;
580 ' DJNZ LUS1           ;IF B <> 0 THEN LUS1
590 ' POP HL
600 ' POP BC
610 ' RET                 ;return
620 ' ;
630 ' EIND END
640 '
650 REM ==============================
660 REM   Basic-testprogramma
670 REM ==============================
680 REM
690 (160) CLS
700 (100) DEFUSR=49500!
710 (85)  SCREEN 2:OPEN"grp:" AS 1
720 (173) PRESET(25,50)
730 (149) PRINT#1,"MSX-BASIC EN ";
740 (87)  PRINT#1,"MACHINETAAL"
750 (11)  PRESET(60,110)
760 (82)  PRINT#1,"TEST-PROGRAMMA"
770 (19)  PRESET(60,118)
780 (41)  PRINT#1,"ctrl+stop=einde"
790 REM
800 REM   MACHINETAAL-ROUTINE
810 REM   wordt in werking gesteld->
820 (157) Z=USR0(0)
830 (130) END
```

Aan het machinetaal-programma is een Basic-test-programma toegevoegd, dat dienst doet als 'aanroep-programma'. De tekstregels worden op het grafische scherm geplaatst (regels 720 - 790), waarna in regel 820 de routine wordt aangeroepen, die de kleuren met regelmatige tussenpozen wijzigt. Het gevolg is dat de tekst afwisselend verschijnt (witte letters op een blauwe achtergrond) en 'verdwijnt' (het bovenste gedeelte van het scherm wordt zwart).

# 6. Testprogramma's

### *6.1 Het kontrole-som-programma*

Alle Basic-programma's in dit hoofdstuk zijn voorzien van 'kontrole-getallen', getallen die achter de regelnummers tussen haakjes zijn geplaatst, bijvoorbeeld 100 (139) Z=USR0(0).
Getal 100 is het regelnummer; getal 139 is het kontrolegetal. Bij het invoeren van een programma moet dit kontrolegetal ***niet*** ingetypt worden. Men toetst het regelnummer in en daarna de programmatekst, die zich achter het kontrolegetal bevindt. Het kontrolegetal stelt de gebruiker in staat om tijdens de invoer van een Basic-programma te kontroleren of de regel korrekt is overgenomen. Die kontrole is alleen mogelijk, wanneer het 'checksum' of het kontrole-som-programma in de computer is geladen!

Het programma 'checksum' bevat een machinetaal-routine, die tijdens de invoer van Basic-programma-regels een tel-procedure uitvoert, die uiteindelijk resulteert in een som-getal, een getal dat tussen de waarden 1 en 255 zal liggen. Dit kontrole-som-getal wordt na het indrukken van de Enter-toets (een handeling die aangeeft dat de invoer van een programma-regel beëindigd is) op het beeldscherm afgedrukt, op de plaats waar zich normaal gesproken de tekst van funktietoets 2 bevindt.

Wanneer men de regel korrekt heeft overgenomen zal dit getal gelijk moeten zijn aan de waarde die zich in de betreffende regel tussen haakjes bevindt! Is dat niet het geval, dan wijst dat op een foute invoer.
Het 'checksum'-programma kontroleert de gekodeerde Basic-regel en negeert daarbij de spaties in de programma-regel, hetgeen betekent dat men de tekst mag invoeren op de wijze die men gewend is. Men kan de tekst zonder spaties invoeren, men mag een vraagteken gebruiken voor het PRINT-kommando en men mag

kleine letters gebruiken voor de diverse Basic-instrukties.

De tekst in DATA-regels en de tekst die tussen aanhalingstekens is geplaatst (die in de vorm van ASCII-kodegetallen in de gekodeerde Basic-regel is opgeslagen) moet letterlijk worden overgenomen.

Regels die met een 'REM'-instruktie beginnen (dus ook de regels die assembler-instrukties bevatten) worden door het kontrole-programma niet in behandeling genomen. Desgewenst kan men de tekst na de REM-instruktie weglaten.

Het 'checksum'-programma bevat een ingebouwde kontrole-routine, die tijdens het intypen van de machinetaal-data kontroleert of de gegevens juist zijn ingevoerd. Wanneer de invoer korrekt is gelopen zal het programma in het geheugen worden geladen vanaf adres 56730 (een waarde die gekozen is om een probleemloze invoer in iedere MSX-computer mogelijk te maken). Op het beeldscherm zal de tekst

```
PROGRAMMA GELADEN VANAF ADRES 56730
```

verschijnen.

Vanaf dat moment wordt na het intypen van een (genummerde) Basic-programma-regel een kontrole-getal op het scherm geplaatst.

```
100 REM ================================
110 REM          C-H-E-C-K-S-U-M
120 REM
130 REM          W. DUZIJN, 1987
140 REM ================================
150 REM
160 REM
170 SCREEN0:COLOR1,14,14
180 IF PEEK(&H2D) THEN WIDTH 70: ELSE WIDTH 38
190 CLS:START=56730!:PRINT"EVEN GEDULD"
200 SOM=0
210 FOR I=0 TO 199
220 READ A$:A=VAL("&H"+A$)
230 IF A<256 THEN SOM=SOM+A:POKE START+I,A:
    ELSE IF SOM<> A THEN 310 ELSE SOM=0:I=I-1
```

```
240 NEXT
250 CLS:BEEP:PRINT"CHECKSUM":PRINT
260 PRINT"PROGRAMMA GELADEN VANAF ";
270 IF PEEK(&H2D)<>0 THEN PRINT:PRINT
280 PRINT"ADRES";START
290 DEFUSR=START:Z=USR(0)
300 NEW
310 POKE &HF6B5,PEEK(&HF6A3)
320 POKE&HF6B6,PEEK(&HF6A4)
330 BEEP:CLS:PRINT"datafout->":LIST.
340 REM
350 REM CHECKSUM-machinetaal-data
360 REM
370 DATA CD,49,DE,21,DB,FD.36,C3,1,AA,DD,23,71,23,70,C9,
    21,5E,F5,7E,FE,30,DA,33,DE,FE,3A,30,7C,21,1F,F4,97,
    4F,F5,7E,FE,23,20,9,F1,3E,C9,32,DB,FD,C3,33,DE,FE,0
    ,28,41,FE,3A,20,9,8,23,7E,FE,8F,28,36,8,2B,FE,2087
380 DATA 8F,28,30,FE,20,20.4,23,7E,18,E3,6,1,FE,E,20,2,6
    ,3,FE,1C,20,2,6,3,FE,1D.20,2,6,5,FE,1F,20,2,6,9,B9,
    38,1,3C,4F,F1,81,F5,23,7E,10,F4,18,BB,F1,11,8F,F8,1,
    64,0,CD,3A,DE,1,A,0,CD,3A,DE,15FD
390 DATA 1,1,0,CD,3A,DE,6,A,3E,20,12,13,10,FA,CD,C0,0,18
    ,3,CD,55,DE,CD,CF,0,C9,B9,38,4,91,4,18,F9,F5,3E,30,
    80,12,13,F1,C9,21,8F,F8.11,61,DE,1,10,0,ED,B0,C9,11,
    8F,F8,21,61,DE,1,10,0,ED,B0,C9,1ACC,END
```

## *6.2 De logische bewerkingen AND, OR en XOR*

In hoofdstuk 1.5 wordt verwezen naar dit test-programma. Het verdient aanbeveling eerst dit hoofdstuk door te nemen voordat men tot het uittesten van het programma overgaat.

Na RUNnen van dit programma zal er een 'menu' op het scherm verschijnen, dat de gebruiker vier keuze-mogelijkheden biedt: testen van de AND-, OR- of XOR-instruktie of beëindiging van het programma. Wanneer men voor het testen van een instruktie heeft gekozen zal het programma vragen welk getal er in de akku moet worden geplaatst. Het begrip 'akku' wordt gebruikt om er op te wijzen dat binnen een machinetaal-programma logische bewerkingen altijd verlopen via het A-register of de akkumulator. Wanneer men

het eerste getal heeft ingevoerd zal gevraagd worden naar een tweede getal, waarmee de logische bewerking kan worden uitgevoerd.

De ingevoerde getallen en het resultaat van de logische bewerking worden in binaire vorm op het scherm afgeheeld, zodat de werkwijze van de betreffende instrukties (het hoog en laagmaken van afzonderlijke bits in een hyte) kan worden bestudeerd.

```
100 REM ==============================
110 REM   AND-OR-XOR-INSTRUKTIES
120 REM   test-programma
130 REM   logische bewerkingen
140 REM ==============================
150 (160) CLS
160 (94)  KEY OFF:ON ERROR GOTO 150
170 (14)  IN$="-instruktie testen"
180 (21)  PRINT"MENU:":PRINT
190 (20)  PRINT"1=AND";IN$
200 (146) PRINT
210 (227) PRINT"2=OR";IN$
220 (146) PRINT
230 (60)  PRINT"3=XOR";IN$
240 (146) PRINT
250 (178) PRINT"4=EINDE"
260 REM
270 (82)  PRINT:INPUT"Uw keuze is";Z
280 (60)  IF Z<1 OR Z>4 THEN RUN ELSE
          IF Z=4 THEN KEY ON:CLS:END
290 (136) ON Z GOTO 300,310,320
300 (157) Z$="AND":GOTO330
310 (107) Z$="OR":GOTO330
320 (174) Z$="XOR"
330 (144) CLS:PRINT"WELK GETAL ";
340 (137) INPUT"IN DE AKKU";A
350 (181) GOSUB 590:A1=A:A1$=A$:PRINT
360 (184) PRINT"GETAL 2 VOOR ";Z$;
370 (25)  INPUT"-BEWERKING";A
380 (234) GOSUB 590:B1=A:B1$=A$
390 (160) CLS
400 (153) PRINT"LOGISCHE BEWERKINGEN:"
```

```
410 (29)  LOCATE ,7
420 (157) PRINT"Getal-1: ";
430 (61)  PRINTA1;TAB(20);A1$
440 (158) PRINT"Getal-2: ";
450 (63)  PRINTB1;TAB(20);B1$
460 (154) PRINTSTRING$(35,"-")
470 (208) IF Z=1 THEN A=A1 AND B1
480 (210) IF Z=2 THEN A=A1 OR B1
490 (212) IF Z=3 THEN A=A1 XOR B1
500 (236) GOSUB 590:C1=A:C1$=A$
510 (254) PRINTMID$(STR$(A1),2,3);" ";
520 (209) PRINTZ$;B1;"=";C1;TAB(20);
530 (195) PRINTC1$:PRINT:PRINT
540 (113) INPUT"verdergaan J/N";Y$
550 (9)   IF Y$="N" OR Y$="n" THEN RUN
560 (45)  PRINT:PRINT"GETAL 1:";A1
570 (158) ON ERROR GOTO 150:GOTO360
580 REM
590 REM   SUBROUTINE->
600 REM
610 REM   A wordt omgezet in binaire
620 REM   string van 8 bits->
630 (74)  A$=BIN$(A):L=LEN(A$)
640 (219) A$=STRING$(8-L,"0")+A$
650 (143) RETURN
```

### *6.3 De bit-manipulatie-instrukties BIT, SET en RES*

De instrukties BIT, SET en RESet worden besproken in de hoofdstukken 1.5 en 1.6E. Het wordt daarom aanbevolen deze hoofdstukken te raadplegen.

Het test-programma laat in binaire vorm de werking van deze bit-instrukties zien. Nadat men uit het 'menu' dat op het scherm verschijnt een keus heeft gemaakt zal om de invoer van een getal worden gevraagd. Dat getal zal in binaire vorm worden afgebeeld op het scherm en er wordt gevraagd welk bit in de byte bewerkt moet worden (dus: getest, geset = hoog gemaakt of gereset = laag gemaakt).

Na iedere bewerking zal de vraag "verdergaan J/N" verschijnen. Is het antwoord "N" of "n" dan wordt het programma opnieuw geRUNd. Beëindiging van het programma is alleen mogelijk door te kiezen voor 'menu'-mogelijkheid nummer 4.

```
100 REM ================================
110 REM   Z80-BIT-SET-RES-INSTRUKTIES
120 REM
130 REM   Basic-testprogramma
140 REM ================================
150 REM
160 (6)   CLS:COLOR 1,14:WIDTH 38
170 (32)  T=0:ZERO$="?":BIT$="?"
180 (54)  KEY OFF:ON ERROR GOTO 110
190 (210) STOP ON:ON STOP GOSUB 430
200 (170) PRINTSTRING$(35,"=")
210 (250) PRINT:PRINT"BIT: ";
220 (134) PRINT"is een bit 0 of 1?"
230 (58)  PRINT"SET: ";
240 (255) PRINT"maak een bit hoog (1)"
250 (56)  PRINT"RES: ";
260 (231) PRINT"maak een bit laag (0)"
270 (146) PRINT
280 (170) PRINTSTRING$(35,"=")
290 (180) LOCATE 0,9:PRINT"MENU:"
300 (146) PRINT
310 (68)  PRINT"1=BIT-instruktie testen"
320 (146) PRINT
330 (81)  PRINT"2=SET-instruktie testen"
340 (146) PRINT
350 (80)  PRINT"3=RES-instruktie testen"
360 (146) PRINT
370 (127) PRINT"4=EINDE":PRINT
380 (55)  PRINT"Uw keuze is";
390 (64)  Z=VAL(INPUT$(1))
400 (164) IF Z<1 OR Z>4 THEN RUN
410 (189) IF Z=4 THEN KEY ON:CLS:END
420 (74)  ON Z GOTO 440,450,460
430 (139) RUN
440 (47)  Z$="BIT":Z1$="GETEST":GOTO470
450 (230) Z$="SET":Z1$="GESET":GOTO470
```

```
460 (209) Z$="RES":Z1$="GERESET"
470 (160) CLS
480 (44)  INPUT"WELK GETAL IN DE AKKU";A
490 REM   A wordt omgezet in binaire
500 REM   string van 8 bits->
510 (74)  A$=BIN$(A):L=LEN(A$)
520 (168) A$=STRING$(8-L,"0")+A$:PRINT
530 (98)  PRINT"Binaire Waarde:";
540 (94)  PRINT TAB(20);A$
550 (82)  PRINT TAB(20);BYTE$
560 (27)  IF Z<>1 THEN 600
570 (78)  PRINT"Zero-vlag=";ZERO$;
580 (174) PRINTTAB(20);"Test-Bit=";BIT$
590 (78)  FOR I=0 TO 13:PRINT:NEXT
600 (67)  COLOR 1,14,14
610 (10)  PRINT:PRINT"verdergaan J/N? ";
620 (124) Y$=INPUT$(1)
630 (9)   IF Y$="N" OR Y$="n" THEN RUN
640 (94)  PRINT:PRINT
650 (212) BYTE$=STRING$(8,32)
660 (50)  PRINT"WELK BIT MOET WORDEN ";
670 (8)   PRINTZ1$;:INPUT"";I
680 (24)  PRINT:PRINTZ$;
690 (127) COLOR 14,14,14
700 (66)  PRINTUSING"##";I;:PRINT",A"
710 (92)  PRINT:ZERO$="0 (NZ)":I=8-I
720  REM
730 (120) MID$(BYTE$,I,1)="XQ"
740 (198) IF Z<>1 THEN 770
750 (231) BIT$=MID$(A$,I,1)
760 (72)  IF BIT$="0" THEN ZERO$="1 (Z)"
770 (27)  IF Z=2 THEN MID$(A$,I,1)="1"
780 (27)  IF Z=3 THEN MID$(A$,I,1)="0"
790 REM
800 (30)  A=VAL("&B"+A$) 'string->getal
810 (165) PRINT"AKKU-INHOUD:";A
820 (213) ON ERROR GOTO 300:GOTO510
```

### *6.4 De komplement- en de twee-komplement-(Neg)-bewerkingen*

Voor een bespreking van de CPL (komplement) en NEG (twee-komplement) bewerkingen wordt verwezen naar hoofdstuk 2.9.

Het test-programma laat zien wat het binaire resultaat is van deze twee bewerkingen. Wanneer in het programma wordt gesproken over de 'twee-komplement'-bewerking dan wordt gedoeld op de Z80-instruktie 'NEG', die een getal een negatieve waarde geeft.

Duidelijk ziet men hoe het teken of 'sign'-bit van een getal (bit 7) hoog (1) wordt wanneer het getal negatief is. Omdat ook de decimale waarden worden afgedrukt kan men precies zien welke getallen door de computer als negatieve getallen en welke getallen als positieve getallen worden opgevat.

```
100 REM ================================
110 REM   TWEE-COMPLEMENT
120 REM
130 REM   Basic-testprogramma
140 REM ================================
150 REM
160 (244) CLS:KEY OFF:STOP ON
170 (254) ON STOP GOSUB 600
180 (170) PRINTSTRING$(35,"=")
190 (106) PRINT"CPL=complement";
200 (223) PRINT"-inverteer"
210 (241) PRINT"Twee-Complement";
220 (252) PRINT"=CPL+INC"
230 (228) PRINT"INC=increment-verhoog"
240 (170) PRINTSTRING$(35,"=")
250 (174) LOCATE 0,7:T$=" testen"
260 (97)  PRINT"1=CPL-instruktie";T$
270 (146) PRINT
280 (219) PRINT"2=Twee-Complement";T$
290 (146) PRINT
300 (126) PRINT"3=EINDE":PRINT
310 (3)   PRINT:PRINT"Uw Keus Is ";
320 (64)  Z=VAL(INPUT$(1))
330 (58)  IF Z<1 OR Z>3 THEN RUN ELSE
          IF Z=3 THEN CLS:KEY ON:END
```

```
340 (249) Z$(1)="CPL"
350 (120) Z$(2)="TWEE-COMPLEMENT"
360 (247) Z$=" en INC A (1 toevoegen)"
370 (155) CLS:Z1$=" BEWERKING:"
380 (78)  PRINT"WELK GETAL IN ";
390 (241) INPUT"DE AKKU";A
400 (176) GOSUB 610:PRINT
410 (36)  PRINT"Dat is binair: ";A$
420 (83)  IF MSB$="1" THEN PRINT
          :PRINT"Negatieve waarde:";A-256
430 (134) PRINT:PRINT"NA ";Z$(Z);Z1$
440 (243) IF Z=2 THEN PRINT Z$(1);Z$
450 (141) PRINT:FOR I=1 TO 8
460 (71)  IF MID$(A$,I,1)="0" THEN MID$(A$,I,1)="1"
          ELSE MID$(A$,I,1)="0"
470 (132) NEXT
480 (30)  A=VAL("&B"+A$)
490 (58)  IF Z=2 THEN A=A+1
500 (108) GOSUB 610:SIGN$=MSB$
510 (188) PRINT"Decimale waarde:";
520 (248) PRINT"+";MID$(STR$(A),2,3);
530 (74)  IF MSB$="1" THEN PRINT " of ";A-256;
540 (94)  PRINT:PRINT
550 (190) PRINT"Dat is binair: ";
560 (196) PRINT A$:PRINT
570 (193) PRINT"Sign-vlag=";SIGN$:PRINT
580 (79)  PRINT"druk op Enter-toets"
590 (100) A$=INPUT$(1)
600 (139) RUN
610 REM   subroutine->
620 REM   --------------------------
630 REM   A wordt omgezet in binaire
640 REM   string van 8 bits
650 REM   --------------------------
660 (248) IF A=256 THEN A=0
670 (74)  A$=BIN$(A):L=LEN(A$)
680 (219) A$=STRING$(8-L,"0")+A$
690 (113) MSB$=LEFT$(A$,1)
700 REM   MSB=Most Significant Bit
710 (143) RETURN
```

### *6.5 De schuif-instrukties SRL, SRA en SLA*

De Z80-schuif-instrukties worden besproken in hoofdstuk 2.11a. Het test-programma laat zien wat er gebeurt wanneer een getal wordt onderworpen aan een 'schuif-bewerking'.

Men kan zien dat een verschuiving naar links een vermenigvuldiging van het getal met de waarde 2 tot gevolg heeft, en dat een verschuiving naar rechts een deling bewerkstelligt. Omdat de carry-vlag door deze instrukties wordt beïnvloed wordt de inhoud van het carry-bit getoond, zodat men kan zien wanneer de bewerking een 'carry'-situatie tot gevolg heeft (inhoud bit = 1) en wanneer de bewerking een 'not-carry'-situatie in het leven roept (inhoud carry-bit = 0).

Een bijzondere schuif-opdracht is de instruktie SRA. Deze instruktie laat het teken-bit (bit 7) intakt, zodat bewerkingen met negatieve getallen mogelijk zijn. Dat laatste is overigens alleen van belang wanneer men volgens de twee-komplement-methode wil rekenen; een methode waar de beginnende programmeur weinig mee te maken zal krijgen. Voor hem zijn vooral de mogelijkheid tot verplaatsing van bits en de deel- en vermenigvuldig-mogelijkheden interessant.

```
100 REM ==============================
110 REM   Z80-SCHUIF-INSTRUKTIES
120 REM
130 REM   Basic-testprogramma
140 REM ==============================
150 REM
160 REM   SRL:Shift Right Logical
170 REM   SRA:Shift Right Arithmetic
180 REM   SLA:Shift Left Arithmetic
190 REM
200 (239) SCREEN 0:COLOR 1,14,14
210 (160) STOP ON:ON STOP GOSUB 380
220 (250) RI$="RECHTS:"
230 (50)  LSB$="?":MSB$="?"
240 (14)  IN$="-instruktie testen"
250 (73)  KEY OFF:T=0
260 (21)  PRINT"MENU:":PRINT
```

```
270 (49)  PRINT"1=SRL";IN$
280 (146) PRINT
290 (39)  PRINT"2=SRA";IN$
300 (146) PRINT
310 (34)  PRINT"3=SLA";IN$
320 (146) PRINT
330 (178) PRINT"4=EINDE"
340 (82)  PRINT:INPUT"Uw Keus Is";Z
350 (160) CLS
360 (60)  IF Z<1 OR Z>4 THEN RUN ELSE
          IF Z=4 THEN CLS:KEY ON:END
370 (135) ON Z GOTO 390,400,620
380 (139) RUN
390 (15)  Z$="SRL":GOTO660
400 (154) Z$="SRA"
410 (201) PRINT"DE SRA-INSTRUKTIE ";
420 (105) PRINT"VERANDERT HET"
430 (155) PRINT"TEKEN-BIT ";
440 (211) PRINT"(bit 7) NIET"
450 (62)  PRINT"IS BIT 7 HOOG (1)"
460 (70)  PRINT"d.w.z. het getal ";
470 (28)  PRINT"is negatief!"
480 (121) PRINT"DAN BLIJFT HET HOOG"
490 (70)  PRINT"d.w.z. het getal ";
500 (189) PRINT"blijft negatief!"
510 (160) PRINT"ER MOETEN HIER ";
520 (13)  PRINT"DAAROM OFWEL "
530 (226) PRINT"POSITIEVE GETALLEN"
540 (7)   PRINT"(0 t/m 127)"
550 (46)  PRINT"OFWEL NEGATIEVE ";
560 (38)  PRINT"GETALLEN"
570 (101) PRINT"(-128 t/m -1)"
580 (113) PRINT"WORDEN INGEVOERD!"
590 (48)  PRINT:A$=INPUT$(1)
600 (160) CLS
610 (47)  GOTO660
620 (129) Z$="SLA":RI$="LINKS:"
630 REM
640 REM   HOOFDPROGRAMMA->
650 REM
660 (238) PRINT"WELK GETAL MOET ";
```

```
670 (51)  INPUT"WORDEN BEWERKT";A
680 (148) IF Z<>2 THEN 720
690 (186) IF A>127 THEN PRINT"Overflow":BEEP:GOTO660
700 (36)  IF A<0 THEN A=A+256:
          IF A<128 THEN PRINT"Overflow": BEEP:GOTO660
710 (147) IF A>127 THEN MSB$="1"
720 (197) IF T=0 THEN 760
730 (218) PRINTUSING"#. ";T;
740 (102) PRINT"Waarde is nu:";
750 (186) IF Z=2 AND MSB$="1"THENPRINT A-256 ELSEPRINT A
760 REM   A wordt omgezet in binaire
770 REM   string van 8 bits->
780 (74)  A$=BIN$(A):L=LEN(A$)
790 (168) A$=STRING$(8-L,"0")+A$:PRINT
800 (199) PRINT"BINAIR:";TAB(8);A$
810 (239) IF Z=3 THEN BI$=MSB$ ELSE BI$=LSB$
820 (171) PRINT"uitgeschoven bit=";
830 (230) PRINTBI$;" -> CARRY-Vlag=";
840 (26)  PRINTBI$:T=T+1
850 (207) Y$=INPUT$(1):PRINT:PRINTZ$;
860 (9)   PRINT"-VERSCHUIVING NAAR ";
870 (253) PRINTRI$:MSB$=LEFT$(A$,1)
880 REM   MSB=Most Significant Byte
890 (113) LSB$=RIGHT$(A$,1)
900 REM   LSB=Least Significant Byte
910 (254) IFZ=1THENA$="0"+LEFT$(A$,7)
920 (145) IFZ=2THENA$=MSB$+LEFT$(A$,7)
930 (1)   IFZ=3THENA$=RIGHT$(A$,7)+"0"
940 (30)  A=VAL("&B"+A$) 'BIN->DEC
950 (24)  IF T<9 THEN 730 ELSE RUN
```

## *6.6 De roteer-instrukties RL, RR, RLC en RRC*

Deze instrukties worden besproken in hoofdstuk 2.11b. Het test-programma toont een 'menu' met 5 keuze-mogelijkheden. Kiest men voor het demonstreren van de werking van een roteer-instruktie dan zal worden gevraagd welk getal er moet worden bewerkt.

Het ingevoerde getal zal in binaire vorm worden afgebeeld, tesamen met het ingeschoven bit en de inhoud van het carry-bit van het vlag-

register: de carry-vlag. Na elke bewerking wacht het programma op het indrukken van een willekeurige toets om voortzetting mogelijk te maken.

Wanneer een cyclische rotatie wordt getest (RLC en RRC) zullen acht bewerkingen worden uitgevoerd (tenzij men door het indrukken van de Ctrl en Stop-toetsen terugkeert naar het menu). Na deze acht bewerkingen is de oorspronkelijke inhoud hersteld. Wanneer een 'gewone' rotatie wordt getest (RR en RL) zullen negen bewerkingen worden uitgevoerd. Men zal kunnen konstateren dat in dit geval de oorspronkelijke inhoud na negen bewerkingen wordt hersteld.

De toelichtingen in de REM-regels hoeven niet ingetypt te worden, de REM regels zelf echter wel, omdat het voor kan komen dat binnen het programma een sprong wordt gemaakt naar een dergelijke regel.

```
100 REM   ===========================
110 REM   Z80-ROTEER-INSTRUKTIES
120 REM
130 REM   Basic-testprogramma
140 REM   ===========================
150 REM
160 REM   RL:Roteer Links
170 REM   RR:Roteer Rechts
180 REM   RLC:Roteer Links Cyclisch
190 REM   RRC:Roteer Rechts Cyclisch
200 REM
210 (189) SCREEN 0:COLOR 1,14
220 (200) STOP ON:ON STOP GOSUB 420
230 (215) WIDTH 38
240 (143) C$="0":CY$=" CYCLISCH:"
250 (219) L$=" LINKS":R$=" RECHTS"
260 (14)  IN$="-instruktie testen"
270 (185) KEY OFF
280 (21)  PRINT"MENU:":PRINT
290 (222) PRINT"1=RL";IN$
300 (146) PRINT
310 (229) PRINT"2=RR";IN$
320 (146) PRINT
330 (35)  PRINT"3=RLC";IN$
```

```
340 (146) PRINT
350 (42)  PRINT"4=RRC";IN$
360 (146) PRINT
370 (179) PRINT"5=EINDE"
380 (82)  PRINT:INPUT"Uw Keus Is";Z
390 (78)  R=9:T=0:BI$="?"
400 (62)  IF Z<1 OR Z>5 THEN RUN ELSE
          IF Z=5 THEN CLS:KEY ON:END
410 (103) ON Z GOTO 430,450,470,480
420 (139) RUN
430 (25)  Z$="RL":RI$=L$:CY$=":"
440 (27)  R=10:GOTO 490
450 (37)  Z$="RR":RI$=R$:CY$=":"
460 (27)  R=10:GOTO 490
470 (175) Z$="RLC":RI$=L$:GOTO490
480 (252) Z$="RRC":RI$=R$
490 (160) CLS
500 (238) PRINT"WELK GETAL MOET ";
510 (51)  INPUT"WORDEN BEWERKT";A
520 (7)   IF T=0 THEN 570
530 (230) PRINTUSING"#: ";T;
540 (168) PRINT"Waarde is nu:";A
550 REM   A wordt omgezet in binaire
560 REM   string van 8 bits->
570 (146) PRINT
580 (74)  A$=BIN$(A):L=LEN(A$)
590 (219) A$=STRING$(8-L,"0")+A$
600 REM
610 (108) PRINT"BINAIR:  ";A$
620 (48)  PRINT"ingeschoven bit=";
630 (230) PRINTBI$;TAB(22);
640 (245) PRINT"CARRY-Vlag=";C$
650 (165) T=T+1:PRINTSTRING$(36,"_")
660 (73)  Y$=INPUT$(1):PRINT
670 (174) IF T=R THEN RUN ELSE PRINTZ$;
680 (62)  PRINT"-ROTATIE NAAR";RI$;CY$
690 (61)  PRINT:MSB$=LEFT$(A$,1)
700 REM   MSB=MOST SIGNIFICANT BYTE
710 (113) LSB$=RIGHT$(A$,1)
720 REM   LSB=LEAST SIGNIFICANT BYTE
730 (44)  RE$=RIGHT$(A$,7)
```

```
740 (41)  LI$=LEFT$(A$,7)
750 (9)   IF Z=1 THEN BI$=C$:A$=RE$+C$:C$=MSB$
760 (7)   IF Z=2 THEN BI$=C$:A$=C$+LI$:C$=LSB$
770 (75)  IF Z=3 THEN BI$=MSB$:A$=RE$+MSB$:C$=MSB$
780 (71)  IF Z=4 THEN BI$=LSB$:A$=LSB$+LI$:C$=LSB$
790 (30)  A=VAL("&B"+A$) 'BIN->DEC
800 (173) GOTO530
```

## *6.7 De stapel en het vlag-register*

In het hiernavolgende test-programma wordt de gebruiker de mogelijkheid geboden met behulp van een aantal stapel-instrukties (zie hoofdstuk 2.17) de inhoud van het vlag-register (zie figuur 1.5c) te bestuderen na een rekenkundige of logische bewerking.

Het programma bestaat uit drie delen:

a. een machinetaal-routine, waarvan de regels 155 en 160 gewijzigd kunnen worden;

b. het Basic-hoofd-programma, dat de inhoud van het stapel-geheugengebied en van het Vlag-register toont;

c. een Basic-subroutine, waarin een machinetaal-routine is opgenomen, die onveranderlijk is.

Met behulp van een Z80-laad-instruktie wordt de stapel verplaatst naar een vrij gedeelte in het RAM-geheugen: regel 140. (Wanneer men niet werkt met het FLASH-assembler/disassembler-programmapakket moet men er zorg voor dragen dat het gebied vanaf lokatie 49500 buiten het bereik van de Basic-interpreter wordt geplaatst!) Om te voorkomen dat de machine in de war raakt wordt eerst de oorspronkelijke inhoud van het stapel-register in het geheugen opgeslagen (regel 135), zodat een herstel bij terugkeer mogelijk is (regel 180).

De inhoud van registerpaar HL wordt bij wijze van voorbeeld op de stapel geplaatst, waarna in de regels 155 en 160 een rekenkundige bewerking wordt uitgevoerd, die een beïnvloeding van het vlag-register tot gevolg heeft.

Het Basic-programma laat de inhouden van het stapel-gebied (de lokaties op de adressen 50000 - 49996) zien na de uitvoering van de machinetaal-routine. Omdat in regel 165 de inhoud van het A-register en de inhoud van het F- of vlag-register in het geheugen worden gezet is door uitlezing van de betreffende geheugenlokatie een binaire weergave van de inhoud van het vlag-register mogelijk. Die inhoud wordt op het scherm afgebeeld.

De in het boek afgedrukte machinetaal-routine telt de waarden 10 en 1 bij elkaar op, hetgeen een positief eindresultaat tot gevolg heeft. Het tekenbit (bit 7) van het vlag-register zal derhalve 'nul' zijn. Omdat het resultaat niet gelijk is aan 'nul' zal het zero-bit eveneens 'nul' zijn. Door de waarden in regel 155 en 160 te variëren kan de werking van het vlag-register worden bestudeerd. Zo zal bijvoorbeeld de zero-vlag hoog (1) zijn wanneer men zowel in regel 155 als in regel 160 de getalwaarde 'nul' invoert.

Wil men andere rekenkundige en logische bewerkingen testen, dan zal de 'ADD,getal'-instruktie moeten worden vervangen door een overeenkomstige instruktie (bijvoorbeeld 'AND 12' of 'SUB 100').

Wil men alleen de ADD-instruktie testen dan kan men gebruik maken van de Basic-subroutine (regels 715 en verder). De vraag:

```
MOET HET ZELFINGEVOERDE MACHINETAAL-PROGRAMMA
WORDEN GETEST
```

moet in dat geval met 'N' (nee) worden beantwoord. Het programma zal nu verzoeken twee getallen in te voeren, die het bij elkaar op kan tellen. Het resultaat van de bewerking wordt op het scherm afgebeeld.

```
100 ' ;=================================
105 ' ;    STACK of STAPEL
110 ' ;    en het vlag-register
115 ' ;    Machinetaal-programma
120 ' ;=================================
125 ' ;
130 ' ORG 49500
135 ' START LD (MEM1),SP
140 ' LD SP,50000
```

```
145 ' LD HL,&H1250  ;variabel
150 ' PUSH HL       ;save HL
155 ' LD A,10       ;variabel
160 ' ADD A,1       ;variabel
165 ' PUSH AF       ;save A+F-registers
170 ' POP AF
175 ' POP HL
180 ' LD SP,(MEM1)
185 ' EIND RET
190 ' MEM1 DS 2
195 ' END
200 '
205 REM ================================
210 REM   Basic-testprogramma
215 REM ================================
220 REM
225 (165) CLS:KEY OFF:WIDTH 39
230 (194) PRINT"MOET HET ZELF INGEVOER";
235 (173) PRINT"DE MACHINETAAL-"
240 (136) PRINT:PRINT"PROGRAMMA ";
245 (183) PRINT"WORDEN GETEST?"
250 (2)   PRINT:INPUT"J/N";B$
255 (32)  IF B$="J" OR B$="j" THEN 455
260 (104) GOSUB 720:CLS:DIM MT$(12)
265 (47)  PRINT"HET BASIC-PROGRAMMA ";
270 (82)  PRINT"TEST HET VOLGENDE"
275 (139) PRINT"MACHINE-TAAL-";
280 (135) PRINT"PROGRAMMA:":PRINT
285 (44)  PRINT"START LD (MEM1),SP"
290 (47)  PRINT"LD SP,50000"
295 (98)  PRINT"LD HL,&H1250"
300 (47)  PRINT"PUSH HL ;save HL"
305 (21)  PRINT"LD A,getal1"
310 (146) PRINT"ADD A,getal2 ;tel op
315 (190) PRINT"PUSH AF ;AF op STACK"
320 (79)  PRINT"POP AF"
325 (92)  PRINT"POP HL"
330 (154) PRINT"LD SP,(MEM1)"
335 (230) PRINT"EIND RET"
340 (179) PRINT"MEM1 DS 2"
345 (124) PRINT"END":PRINT
```

```
350 (157) PRINT"GETAL1 WORDT OPGETELD";
355 (79)  PRINT" BIJ GETAL2"
360 (66)  PRINT"HET RESULTAAT WORDT ";
365 (90)  PRINT"VIA DE INSTRUKTIE"
370 (43)  PRINT"'PUSH AF' ";
375 (79)  PRINT"OP DE STAPEL GEZET"
380 (149) PRINT"HET F OF VLAG-REGISTER";
385 (106) PRINT" WORDT UITGELEZEN"
390 (124) Y$=INPUT$(1)
395 (160) CLS
400 (115) INPUT"WAT IS GETAL1";A
405 (65)  IF A>255 THEN 395
410 (1)   POKE 49512!,A:CLS
415 (18)  PRINT"INHOUD AKKU=&H";HEX$(A);
420 (49)  PRINT" -> DECIMAAL:";A
425 (130) PRINT:PRINT"WELK GETAL ";
430 (80)  PRINT"MOET ER BIJ WORDEN"
435 (108) PRINT:INPUT"OPGETELD";A
440 (95)  IF A>255 THEN 425
445 (239) POKE 49514!,A:B$="":GOTO 510
450 REM
455 (160) CLS
460 (176) PRINT"IS HET MACHINETAAL";
465 (123) PRINT"-PROGRAMMA":PRINT
470 (168) PRINT"IN DE COMPUTER ";
475 (7)   PRINT"GELADEN VANAF ":PRINT
480 (186) PRINT"HET ADRES 49500..."
485 (25)  PRINT:INPUT"J/N";Y$
490 (38)  IF Y$<>"J"ANDY$<>"j" THEN 705
495 REM
500 REM   HOOFD-PROGRAMMA->
505 REM
510 (222) DEFUSR0=49500!:Z=USR0(0):T=0
515 (181) RESTORE 535
520 (246) FOR I=1 TO 4:READ R$
525 (53)  R$(I)="Register: "+R$
530 (132) NEXT
535 (39)  DATA H,L,A,F
540 (160) CLS
545 (170) PRINT"STAPEL:":PRINT
550 (196) FOR I=50000! TO 49995! STEP-1
```

```
555 (133) PRINT"ADRES ";
560 (105) PRINTUSING"#####: ";I;
565 (116) P=PEEK(I)
570 (59)  RI$(T)="&H"+HEX$(P)
575 (35)  PRINT RI$(T);TAB(17);P;
580 (24)  PRINT TAB(22);R$(T)
585 (92)  T=T+1:NEXT
590 (232) PRINT:PRINT"Adres 49996 ";
595 (65)  PRINT"bevat de inhoud "
600 (243) PRINT"van het VLAG-";
605 (8)   PRINT"(F)-REGISTER"
610 (64)  A=VAL(RI$(4))
615 (74)  A$=BIN$(A):L=LEN(A$)
620 (219) A$=STRING$(8-L,"0")+A$
625 (116) FLAG$="SZ---V-C"
630 (42)  PRINT:PRINT"INHOUD F:";
635 (59)  PRINT TAB(22);FLAG$
640 (45)  PRINT TAB(22);A$:PRINT
645 (23)  PRINT"S=Sign-Vlag";TAB(16)
650 (179) PRINT"1:negatief getal in A"
655 (45)  PRINT"Z=Zero-Vlag";TAB(16)
660 (105) PRINT"1:resultaat=0
665 (224) PRINT"V=oVerflow-Vlag";TAB(16)
670 (155) PRINT"1:overflow
675 (120) PRINT"C=Carry-Vlag";TAB(16)
680 (72)  PRINT"1:Carry"
685 (195) IF B$<>"" THEN 705
690 (10)  PRINT:PRINT"verdergaan J/N? ";
695 (87)  Y$=INPUT$(1):CLS
700 (23)  IF Y$="J" OR Y$="j" THEN 400
705 (30)  KEY ON:END
710 REM
715 REM ================================
720 REM   Basic-subroutine
725 REM   machinetaal in DATA-regels
730 REM   adres 49512 = inhoud akku
735 REM ================================
740 (150) RESTORE 760
745 (247) FOR I=0 TO 24:READ A$
750 (240) POKE 49500!+I,VAL("&H"+A$)
755 (77)  NEXT:RETURN
```

```
760 (163) DATA ED,73,73,C1,31,50,C3
765 (149) DATA 21,50,12,E5,3E,64,C6
770 (206) DATA 14,F5,F1,E1,ED,7B,73
775 (92)  DATA C1,C9,0,0
```

### *6.8 De uitvoer-instruktie OUT*

Met behulp van de 'OUT'-instruktie kunnen gegevens via de poorten van de MSX-computer naar de randapparatuur (printer, beeldscherm e.d.) worden getransporteerd. De 'OUT'-instruktie wordt besproken in hoofdstuk 2.19 (Z80) en hoofdstuk 3.20 (Basic).

In het testprogramma dat hiernaast staat afgedrukt worden zowel de Z80-instruktie als de Basic-instruktie getest. De machinetaal-routine gaat vooraf aan bet Basic-programma en moet alvorens het Basic-programma wordt geRUNd in de computer worden geladen. (Bezitters van de FLASH-assembler hoeven slechts de 'asmSCR-toets' in te drukken.)

Het Basic-programma vraagt om de invoer van een 'teken' (letter, cijfer of grafisch teken). Wanneer het gewenste teken is ingetoetst wordt gevraagd of het machinetaal-programma of de Basic-subroutine moet worden getest. Wordt gekozen voor Basic dan zal de subroutine in de regel 700 tot en met 800 worden uitgevoerd.

In regel 750 van deze routine wordt het startadres in het video-RAM toegekend aan de variabelen H en L. Daabij wordt bit 6 van de waarde in H hoog gemaakt (om aan te geven dat het om een 'schrijf-operatie' gaat). Eerst wordt de lage waarde (L) naar kommando-poort &H99 geschreven; daarna de hoge waarde (H). Wanneer het adres is doorgegeven kan het ingetoetste teken naar het video-RAM worden geschreven via data-poort &H98. Dat gebeurt in de regels 780 en 790.

De machinetaal-routine werkt volgens hetzelfde principe:

Regels 200 - 240: adres naar kommando-poort &H99 (bit 6 hoog);
Regels 260 - 300: teken naar data-poort &H98.

```
100 ' ;=================================
110 ' ;   GEGEVENS SCHRIJVEN NAAR VRAM
120 ' ;   m.b.v. de Z80-OUT-instruktie
130 ' ;   Machinetaal-programma
140 ' ;=================================
150 ' ;
160 ' ORG 49500
170 ' START DI
180 ' ;maak interrupts onmogelijk
190 ' ;
200 ' LD C,&H99 ;adrespoort in C
210 ' LD HL,0 ;startadres in HL
220 ' SET 6,H ;maak schrijfbit 1
230 ' OUT (C),L
240 ' OUT (C),H
250 ' ;
260 ' LD C,&H98 ;datapoort in C
270 ' LD A,(TEKEN)
280 ' LD B,240 ;aantal tekens
290 ' LUS OUT (C),A
300 ' DJNZ LUS ;IF B<>0 THEN LUS
310 ' ;
320 ' EI ;maak interrupts mogelijk
330 ' RET ;terug naar Basic
340 ' ;
350 ' TEKEN NOP ;Basic-Poke-adres
360 ' EIND END
370 ' ;
380 REM ================================
390 REM   Basic-testprogramma
400 REM ================================
410 REM
420 (147) CLS:KEYOFF
430 (19)  COLOR 1,5
440 (100) DEFUSR=49500!
450 (189) PRINT"TEST-PROGRAMMA":PRINT
460 (121) PRINTSTRING$(37,"="):PRINT
470 (21)  PRINT"OUT-INSTRUKTIE"
480 (94)  PRINT:PRINT
490 REM
500 (199) INPUT"TEKEN";T$:A=ASC(T$)
```

```
510 (164) IF A=1 THEN 500
520 REM   plaats ASCII-waarde van het
530 REM   teken in geheugenplaats
540 REM   49525 = TEKEN ->
550 (87)  POKE 49525!,A
560 REM
570 (181) INPUT"KLEUR";KL:COLOR KL
580 (146) PRINT
590 (43)  PRINT"1. machinetaal of"
600 (104) PRINT"2. Basic":PRINT
610 (134) INPUT"Uw Keus Is";Z
620 (195) IF Z=2 THEN 660 ELSE 640
630 REM   roep machinetaalroutine aan->
640 (7)   Z=USR0(0):GOTO 670
650 REM   roep basicsubroutine aan->
660 (141) GOSUB 760
670 (79)  PRINT"druk op ENTER-toets"
680 (100) A$=INPUT$(1)
690 (183) COLOR 1,14:KEY ON
700 (130) END
710 REM ===============================
720 REM   Gegevens schrijven naar VRAM
730 REM   m.b.v. de Basic-instruktie OUT
740 REM   subroutine->
750 REM ===============================
760 (220) H=VAL("&B01000000"):L=0
770 REM   schrijfbit:bit 6 van H=1
780 (67)  OUT &H99,L:OUT &H99,H
790 (168) FOR I=1 TO 240
800 (66)  OUT &H98,A:NEXT
810 (143) RETURN
```

## *6.9 RAM en video-RAM: blokverplaatsing*

In de hoofdstukken 3.6 (COPY), 3.33 (VPEEK) en 3.34 (VPOKE) worden routines besproken die het data-verkeer tussen het RAM-geheugen en het video-RAM mogelijk maken.

In hoofdstuk 5 werd een van die routines (standaardroutine FILVRM) gebruikt voor het inkleuren van het scherm; in het test-

programma dat we hier bespreken worden twee routines gebruikt die gedeelten van het RAM-geheugen naar het video-RAM kunnen verplaatsen en omgekeerd: de standaardroutines LDIRMV en LDIRVM. Met behulp van die routines wordt een onder Basic opgebouwd tekst-scherm uit het RAM-geheugen gehaald.

Die verplaatsing omvat drie stappen:

a. eerst wordt de momentele inhoud in het RAM-geheugen geplaatst (regels 240 - 300);

b. vervolgens wordt het tekst-scherm in het VRAM geladen (regels 320 - 370);

c. tenslotte wordt (na een korte wachtpauze) de oorspronkelijke inhoud hersteld (regels 480 - 540).

```
100 ' ;================================
110 ' ;   LDIR of BLOKVERPLAATSING
120 ' ;   gegevens worden verplaatst
130 ' ;   van RAM naar videoRAM
140 ' ;   en omgekeerd
150 ' ;   MACHINETAAL-PROGRAMMA
160 ' ;================================
170 ' ;
180 ' ORG 49500
190 ' ;
200 ' BEEP EQU &H00C0
210 ' LDIRMV EQU &H0059
220 ' LDIRVM EQU &H005C
230 ' ;
240 ' START CALL BEEP
250 ' ;verplaats scherminhoud naar
260 ' ;RAM-geheugen->
270 ' LD DE,56000 ;Doel-adres in RAM
280 ' LD HL,0 ;beginadres in VRAM
290 ' LD BC,960 ;lengte van het blok
300 ' CALL LDIRMV ;verplaatsroutine
310 ' ;
320 ' ;verplaats opgeslagen tekst
```

```
330 ' ;in RAM naar VRAM->
340 ' LD DE,0 ;Doel-adres in VRAM
350 ' LD HL,55000 ;beginadres in RAM
360 ' LD BC,960 ;lengte van het blok
370 ' CALL LDIRVM ;verplaatsroutine
380 ' ;
390 ' ;vertragingslus(sen)->
400 ' LD B,5 ;teller
410 ' LUS1 LD DE,&HFFFF
420 ' LUS2 DEC DE ;DE=DE-1
430 ' LD A,D
440 ' OR E ;IF (D OR E) <> 0
450 ' JR NZ,LUS2 ;THEN LUS2
460 ' DJNZ LUS1 ;B=B-1
470 ' ;
480 ' ;herstel oorspronkelijke inhoud
490 ' ;RAM naar VRAM->
500 ' CALL BEEP
510 ' LD DE,0 ;Doeladres in VRAM
520 ' LD HL,56000 ;beginadres in RAM
530 ' LD BC,960 ;lengte van het blok
540 ' CALL LDIRVM ;verplaatsroutine
550 ' ;
560 ' RET ;naar Basic
570 ' EIND END
580 '
590 REM ================================
600 REM   Basic-testprogramma
610 REM ================================
620 REM
630 (147) CLS:KEY OFF
640 (100) DEFUSR=49500!
650 (188) PRINT:PRINT"TEST-PROGRAMMA"
660 (76)  PRINT:PRINT"een toelichten";
670 (11)  PRINT"de tekst kan"
680 (95)  PRINT"naar believen wor";
690 (73)  PRINT"den opgeroepen"
700 (75)  PRINT"in het Basic-programma"
710 (187) PRINT:PRINT"Bijvoorbeeld:"
720 (38)  PRINT"U moet een ge";
730 (16)  PRINT"tal raden (0-9)"
```

```
740 (221) PRINT"Het juiste getal ";
750 (31)  PRINT"stopt het programma"
760 (57)  LOCATE 0,18
770 (13)  PRINT"DEZE TEKST IS NU ";
780 (141) PRINT"OPGESLAGEN":PRINT
790 (171) PRINT"IN HET RAM-GEHEUGEN"
800 REM   de tekst wordt met behulp
810 REM   van een Basic-poke-routine
820 REM   opgeslagen in het RAM
830 REM   let op de traagheid ervan->
840 (135) FOR I=0 TO 960
850 (240) POKE55000!+I,VPEEK(0+I)
860 (132) NEXT
870 (108) CLS:PRINT
880 (236) PRINT"MACHINETAAL EN MSX-";
890 (7)   PRINT"Basic":PRINT
900 (117) PRINT"U moet een getal raden:"
910 (153) PRINT"Toelichting J/N? ";
920 (75)  GOSUB990:PRINT
930 (109) INPUT"Uw Keus Is";A
940 (190) B=INT(RND(TIME)*10)
950 (14)  IF A<>B THEN PRINT"FOUT!":GOTO900:
          ELSE CLS:PRINT"JUIST":KEY ON:END
960 REM
970 REM   MACHINETAAL-ROUTINE
980 REM   wordt in werking gesteld->
990 (241) A$=INPUT$(1):IF A$="J" OR A$="j"
          THEN Z=USR0(0):RETURN:ELSERETURN
```

### *6.10 De interruptverwerker*

De MSX-computer roept 50 maal per sekonde een routine aan op het adres &H38 (zie hoofdstuk 3.15). In deze routine is een 'CALL'-instruktie opgenomen, die het adres &HFD9F in het werkgeheugen aanroept (we noemen dat het aanroepen van een 'hook'). Wanneer we op dat adres een sprongopdracht plaatsen, die naar een zelfgeschreven routine voert, kunnen we bereiken dat deze routine ook vijftig maal per sekonde wordt uitgevoerd. Van die mogelijkheid wordt in het hiernavolgende test-programma gebruik gemaakt om een automatische beweging van een sprite tot stand te

brengen.

In de regels 440 - 460 van het Basic-programma wordt de inhoud van de vijf geheugenplaatsen achter het adres &HFD9F tijdelijk verplaatst naar een geheugengebiedje op adres 50000, zodat de oorpronkelijke inhoud weer hersteld kan worden aan het eind van het programma (regels 560 - 580). Deze procedure is noodzakelijk, omdat de mogelijkheid bestaat dat hookadressen reeds in gebruik zijn!

In de regels 470 en 480 wordt een blok-sprite aangemaakt, die in regel 520 op het screen-1-scherm wordt gezet. De machinetaal-routine die in regel 530 wordt aangeroepen plaatst het startadres van de uit te voeren subroutine in het 'hook'- gebiedje (regels 150 - 200), zodat automatische uitvoer van deze routine mogelijk is. Er wordt daarbij van 'achteren' naar 'voren' gewerkt, om te voorkomen dat een verkeerde sprong wordt gemaakt.

De subroutine (regel 210 en verder) voert twee opdrachten uit:

a. er wordt van 1 tot 50 geteld (regels 220 - 250); pas na 50 tellen wordt opdracht b uitgevoerd:

b. de inhoud van lokatie 6917 van de sprite-attribuut-tabel in het video-RAM wordt met de waarde 8 verhoogd, zodat de sprite een horizontale beweging kan uitvoeren (regels 260 - 300). Als opdracht b is uitgevoerd wordt de teller voor opdracht a weer op 'nul' gezet (regel 310).

```
100 ' ;==================================
110 ' ;   INTERRUPT-TEST
120 ' ;   machinetaal-routine
130 ' ;==================================
140 ' ;
150 ' LD IX,&HFD9F        ;HOOK-adres
160 ' LD BC,SUBR          ;adres SUBR
170 ' LD (IX+1),C
180 ' LD (IX+2),B
190 ' LD (IX+0),205       ;205=CALL
200 ' RET                 ;return
210 ' SUBR PUSH AF        ;bewaar AF
```

```
220 ' LD A,(KON)          ;inhoud KON in A
230 ' INC A               ;A=A+1
240 ' CP 50               ;IF A<>50
250 ' JP NZ,UIT           ;THEN UIT
260 ' LD A,(KON1)         ;inhoud KON1 in A
270 ' ADD A,8             ;A=A+8
280 ' LD (KON1),A         ;inhoud A in KON1
290 ' LD HL,6917          ;x-koordinaat sprite
300 ' CALL &H004D         ;VPOKE 6917,A
310 ' SUB A               ;A=A-A=0
320 ' UIT LD (KON),A      ;inhoud A in KON
330 ' POP AF              ;herstel AF
340 ' RET                 ;return
350 ' KON NOP
360 ' KON1 NOP
370 ' EIND END
380 ' ;
390 REM ================================
400 REM   Basic-aanroepprogramma
410 REM ================================
420 (179) SCREEN 1:CLS
430 (82)  COLOR 1,5,1
440 (188) FOR I=0 TO 4
450 (130) POKE50000!+I,PEEK(&HFD9F+I)
460 (132) NEXT
470 (191) FOR I=0 TO 7
480 (100) A$=A$+CHR$(255):NEXT
490 (166) SPRITE$(1)=A$
500 (54)  LOCATE 6,10
510 (25)  PRINT"INTERRUPT-TEST"
520 (51)  PUTSPRITE 1,(0,78),15,1
530 (77)  DEFUSR0=49500!:Z=USR0(0)
540 (59)  LOCATE 0,20
550 (100) A$=INPUT$(1) 'wacht
560 (158) FOR I=4 TO 0 STEP-1
570 (130) POKE&HFD9F+I,PEEK(50000!+I)
580 (44)  NEXT:COLOR 1,14,14
590 (148) SCREEN 0:END
```

### *6.11 Zoek en vervang een 'token' in een Basic-regel*

In hoofdstuk 3.22 wordt in een klein voorbeeld-programma aangegeven hoe het PRINT-token in een regel vervangen kan worden door een LPRINT-token. Omdat een dergelijke vervanging binnen een Basic-programma praktisch nut kan hebben wordt in dit hoofdstuk een wat uitvoeriger versie afgedrukt.

Het programma wil niet meer zijn dan een voorbeeld-programma, dat laat zien op welke wijze een Basic-programma byte voor byte kan worden onderzocht binnen een machinetaal-routine. Ter vergelijking is een soortgelijke Basic-subroutine opgenomen.

Om het beginadres en het eindadres van het Basic-programma te bepalen wordt in de machinetaal-routine gebruik gemaakt van twee systeem-variabelen (zie de regels 130 en 135). Het token dat moet worden opgezocht wordt uit de geheugenlokatie met het adres 49534 gehaald (regel 140). Daarin werd het geplaatst binnen het Basic-programma met behulp van het kommando 'POKE 49534,P1' (regel 600).

Om te voorkomen dat ieder PRINT-kommando in een regel wordt vervangen spreken we af dat we alleen daar ingrijpen waar een dubbele punt aan het kommando voorafgaat (zie de regels 160 - 170). Treft het programma achter een dubbele punt het PRINT-kommando aan dan zal de verwisselroutine in de regels 205 - 215 in werking worden gesteld. Deze verwisselroutine maakt gebruik van de logische instruktie 'XOR'.

Voordat de zoekroutine in gang wordt gezet stelt het Basic-aanroepprogramma de vraag of uitvoer van de gegevens naar de printer is gewenst. Afhankelijk van het antwoord wordt de waarde 145 of 157 toegekend aan de variabele P1.

***Let op:***
Vergeet niet de machinetaal-routine met behulp van de assembler in het geheugen te laden (funktie-toets 5 voor FLASH-gebruikers)!

```
100 ' ;===================================
105 ' ;   TOKEN-ZOEKPROGRAMMA
110 ' ;   machinetaal-zoek-routine
115 ' ;===================================
120 ' ;
125 ' ORG 49500
130 ' START LD HL,33932  ;beginadres
135 ' LD DE,(&HF6C2)     ;eindadres program
140 ' LD A,(TOKEN)       ;token in akku
145 ' LD B,A             ;token in register B
150 ' ;
155 ' LUS INC HL         ;HL=HL+1
160 ' LD A,(HL)          ;A=PEEK(ADRES)
165 ' CP 58              ;dubbele punt?
170 ' JP NZ,LUS          ;nee, dan volgend adres
175 ' ;
180 ' INC HL             ;HL=HL+1
185 ' LD A,(HL)          ;A=PEEK(ADRES)
190 ' CP B               ;vergelijk met token-kode
195 ' JP NZ,VERDER       ;IF A<>B THEN VERDER
200 ' ;
205 ' XOR 12             ;verwissel token
210 ' ;145 XOR 12 = 157
215 ' ;157 XOR 12 = 145
220 ' LD (HL),A          ;POKE(ADRES),A
225 ' ;
230 ' VERDER CALL &H0020 ;DCOMPR
235 ' JP C,LUS
240 ' ;
245 ' ;                   DCOMPR=standaardroutine,
250 ' ;                   die registerparen DE en HL
255 ' ;                   vergelijkt met elkaar
260 ' ;                   als HL<DE dan C=1 (Carry)
265 ' ;
270 ' RET                ;als DE=HL naar Basic
275 ' ;
280 ' TOKEN NOP          ;adres 49534
285 ' ;                   deze geheugenlokatie wordt
290 ' ;                   in het Basic-programma
295 ' ;                   gebruikt!
300 ' EINDE END
```

```
305 ' ;
310 REM
315 REM ================================
320 REM   ZOEK EEN TOKEN
325 REM
330 REM   Basic-testprogramma
335 REM ================================
340 (39)  SCREEN 0:COLOR 1,7
345 (188) KEY OFF:GOTO 375
350 REM ------------------------
355 REM   TEST-regels->
360 (121) :PRINT"Test-programma":PRINT
365 (250) :PRINT"Einde"
370 REM ------------------------
375 (198) PRINT"Basic routine (1)"
380 (172) PRINT"of"
385 (25)  PRINT"machinetaal-zoek";
390 (99)  PRINT"routine (2)"
395 (146) PRINT
400 (43)  A$="Basic-"
405 (204) INPUT"UW KEUS IS ";A
410 (164) CLS:IF A=1 THEN 430
415 (230) A$="MACHINETAAL-"
420 (51)  IF PEEK(49500!)=33 THEN 430
425 (116) PRINT"MT-routine laden":END
430 (109) PRINT"DIT IS EEN ";
435 (137) PRINTA$;"ZOEK-PROGRAMMA"
440 (146) PRINT
445 (140) IF A=2 THEN 460
450 (12)  PRINT"let op de lange ";
455 (106) PRINT"wachttijd!":GOTO 465
460 (56)  PRINT"let op de snelheid!"
465 (146) PRINT
470 (192) PRINT"VRAAG: ";
475 (148) INPUT"printer-uitvoer J/N";Y$
480 (102) IFY$="J" OR Y$="j" THEN P1=145:P2=157:
          ELSE P1=157:P2=145
485 (224) PRINT:IF A=2 THEN 590
490 REM
495 REM ================================
500 REM   Basic-zoekroutine
```

```
505 REM ================================
510 REM   systeemvariabele &HF6C2 bevat
515 REM   startadres variabelengeheugen
520 REM   dit is tevens het EINDE van
525 REM   het Basic-programma
530 REM
535 (188) EINDE=PEEK(&HF6C2)+256*PEEK(&HF6C3)
540 (174) START=32768!+1164
545 REM   ASCII-waarde dubb. punt=58
550 (119  PRINT"het programma zoekt!!!"
555 (205) TIME=0
560 (143) FOR I=START TO EINDE
565 (5)   IF PEEK(I)=P1 AND PEEK(I-1)=58 THEN POKE I,P2
570 (17)  NEXT:T=TIME:GOTO 610
575 REM
580 REM ================================
585 REM   plaats token in geheugen en
590 REM   roep machinetaal-routine aan
595 REM ================================
600 (144) POKE 49534!,P1:TIME=0
605 (152) DEFUSR0=49500!:Z=USR0(0):T=TIME
610 (160) PRINT"aantal seconden:";T/50
615 (233) PRINT:KEY ON:LIST 355-365
```

### *6.12 Het plaatsen van een punt op het scherm: PSET*

Dit test-programma maakt gebruik van de standaardroutines die in hoofdstuk 3.24 worden besproken. Verder wordt gebruik gemaakt van een routine die de computer in een toestand brengt, waarin wordt gewacht op het indrukken van een toets: CHGET (regel 360).

Wordt de Enter-toets ingedrukt (kode-getal 13) dan zal het Basic-programma worden hervat (regel 390); in het andere geval wordt de x-koördinaat met de waarde 4 verhoogd (regels 400 - 430).

```
100 ' ;================================
110 ' ;   PSET - punt op grafisch scherm
120 ' ;   Machinetaalprogramma
130 ' ;================================
140 ' ;
```

```
150 ' ORG 49500
160 ' INIMLT EQU &H0075
170 ' SETATR EQU &H011A
180 ' SCALXY EQU &H010E
190 ' MAPXYC EQU &H0I11
200 ' CHGET EQU &H009F
210 ' SETC EQU &H0120
220 ' STOP EQU &H00BA
230 ' ;
240 ' START CALL INIMLT ;SCREEN 3
250 ' LD A,I5           ;voorgrondkleur in akku
260 ' CALL SETATR       ;vul kleur in
270 ' ;
280 ' LD BC,20          ;X-koord
290 ' LUS PUSH BC       ;bewaar inhoud BC
300 ' LD DE,50          ;Y-koord
310 ' ;
320 ' CALL SCALXY       ;pas schaal aan
330 ' CALL MAPXYC       ;bepaal PUNTpositie
340 ' CALL SETC         ;zet PUNT op scherm
350 ' ;
360 ' CALL CHGET        ;wacht op toetsindruk
370 ' POP BC            ;herstel inhoud BC
380 ' CP 13             ;Enter-toets ingedrukt?
390 ' RET Z             ;ja, dan RETURN
400 ' INC BC
410 ' INC BC
420 ' INC BC
430 ' INC BC            ;X=X+4
440 ' JP LUS            ;herhaal
450 ' ;
460 ' EIND END
470 ' ;
480 REM ================================
490 REM   Basic-aanroepprogramma
500 REM ================================
510 (100) SCREEN 0:COLOR 1,5,1
520 (62)  WIDTH 35:LOCATE 0,7
530 (185) KEY OFF
540 (134) PRINT"indrukken ENTER-toets"
550 (146) PRINT"beeindigt het programma"
```

```
560 (28)  PRINT"elke andere toets"
570 (4)   PRINT"print een stip"
580 (70)  FOR I=0 TO 3000:NEXT
590 (77)  DEFUSR0=49500!:Z=USR0(0)
600 (44)  COLOR 1,14:WIDTH 38
610 (30)  KEY ON:END
```

### *6.13 STICK en STRIG: joystick en vuurknop*

In de hoofdstukken 3.29 en 3.31 worden de Basic-instrukties STICK en STRIG besproken. Dit test-programma laat zien hoe de informatie die daarin wordt verstrekt gebruikt kan worden om een tekening op het scherm te plaatsen met behulp van de kursor (of de joystick, wanneer de juiste indikatie-waarde in de akku wordt geladen: regel 225).

In de regels 235 tot 300 wordt bekeken welke waarde er door de standaardroutine GTSTCK in de akku werd geplaatst, met behulp van een reeks 'compare'-instrukties. Wordt een van de vier genoemde waarden (1, 3, 5 en 7) gevonden dan wordt de daarbij behorende subroutine aangeroepen. Deze subroutines (die de beweging sturen) bevinden zich aan het eind van het bron-programma in de regels 370 - 500.

De vuurknop-routine, GTTRIG, doet dienst om een terugkeer naar het Basic-programma mogelijk te maken. Wordt de spatiebalk ingedrukt dan zal de waarde 255 in de akku worden geladen. De inhoud wordt in regel 210 getest met behulp van een CP-instruktie. De waarde 255 wordt van de inhoud van de akku afgetrokken. Is het resultaat 'nul' dan wordt de zero-vlag hoog (1). De opdracht 'RET Z' in regel 215 wordt in dat geval uitgevoerd.

```
100 ' ;==================================
105 ' ;    STICK(x) en STRIG(x)
110 ' ;    joystick en vuurknop testen
115 ' ;    machinetaalprogramma
120 ' ;==================================
125 ' ;
130 ' ORG 49500
135 ' INITXT EQU &H006C
```

```
140 ' GTTRIG EQU &H00D8
145 ' GTSTCK EQU &H00D5
150 ' BEEP EQU &H00C0
155 ' CLS EQU &H00C3
160 ' ;
165 ' CALL INITXT          ;SCREEN 0
170 ' LD A,16             ;X-koordinaat
175 ' LD (VARX),A
180 ' LD A,12             ;Y-koordinaat
185 ' LD (VARY),A
190 ' JP CURSOR           ;GOTO CURSOR
195 ' ;
200 ' STRIG LD A,0        ;0=kursortoets
205 ' CALL GTTRIG         ;STRIG(0)
210 ' CP 255              ;ingedrukt?
215 ' RET Z               ;JA, dan naar Basic
220 ' ;
225 ' STICK LD A,0        ;a=kursortoets
230 ' CALL GTSTCK         ;STICK(0)
235 ' CP 1                ;omhoog
240 ' JP NZ,TWEE
245 ' CALL UPY
250 ' JP CURSOR
255 ' TWEE CP 3           ;rechts?
260 ' JP NZ,DRIE
265 ' CALL RIGHTX
270 ' JP CURSOR
275 ' DRIE CP 5           ;omlaag?
280 ' JP NZ,VIER
285 ' CALL DOWNY
290 ' JP CURSOR
295 ' VIER CP 7           ;links?
300 ' JP NZ,STRIG
305 ' CALL LEFTX
310 ' ;
315 ' CURSOR LD A,(VARX)
320 ' LD (&HF3DD),A       ;CSRX
325 ' LD A,(VARY)
330 ' LD (&HF3DC),A       ;CSRY
335 ' ;                    voor verklaring CSRX/CSRY
340 ' ;                    zie LOCATE x,y
```

```
345 ' LD A,219            ;blokje
350 ' RST &H18            ;print het blokje
355 ' CALL BEEP           ;beep+vertraag
360 ' JP STRIG
365 ' ;
370 ' UPY LD A,(VARY)     ;omhoog
375 ' DEC A
380 ' CP 2                ;IF Y<2 THEN
385 ' RET C               ;RETURN
390 ' LD (VARY),A
395 ' RET
400 ' ;
405 ' DOWNY LD A,(VARY)   ;omlaag
410 ' INC A
415 ' CP 23               ;IF Y>22 THEN
420 ' RET NC              ;RETURN
425 ' LD (VARY),A
430 ' RET
435 ' ;
440 ' LEFTX LD A,(VARX)   ;links
445 ' DEC A
450 ' CP 1                ;IF X<1 THEN
455 ' RET C               ;RETURN
460 ' LD (VARX),A
465 ' RET
470 ' ;
475 ' RIGHTX LD A,(VARX)  ;rechts
480 ' INC A
485 ' CP 40               ;IF X>39 THEN
490 ' RET NC              ;RETURN
495 ' LD (VARX),A
500 ' RET
505 ' ;
510 ' VARX NOP
515 ' VARY NOP
520 ' EIND END
525 ' ;
530 REM ===============================
535 REM   Basic-aanroepprogramma
540 REM ===============================
545 REM
```

```
550 (224) CLS:KEY OFF:COLOR 1,5
555 (134) IF PEEK(&H2D)<>0 THEN B=1
560 (141) IF B THEN SETBEEP 1,2
565 (165) PRINT"MET BEHULP VAN ";
570 (148) PRINT"DE CURSOR-TOETSEN"
575 (18)  PRINT:PRINT"KAN EEN REEKS ";
580 (102) PRINT"BLOKJES OP HET "
585 (118) PRINT:PRINT"SCHERM WORDEN";
590 (128) PRINT" GEPLAATST"
595 (113) PRINT:PRINT"HET INDRUKKEN ";
600 (88)  PRINT"VAN DE SPATIE-BALK"
605 (111) PRINT:PRINT"BEEINDIGT ";
610 (98)  PRINT"HET PROGRAMMA"
615 (72)  PRINT:Y$=INPUT$(1)
620 (60)  DEFUSR=49500!:Z=USR0(0)
625 (145) CLS:COLOR 1,14:KEY ON
630 (145) IF B THEN SETBEEP 4,3
635 (130) END
```

### *6.14 Een eenvoudig sprite-programma*

In dit test-programma wordt de kennis die in hoofdstuk 3 werd verstrekt samengebracht in een volledig programma dat:

a. het scherm in de screen-1-mode plaatst (regel 130),

b. de schermkleuren invult (regels 1140 - 1180),

c. een tekst op het scherm plaatst (regels 1200 - 1250),

d. een sprite-patroon definieert (regels 1270 - 1300) en

e. de sprite-attribuut-tabel invult (regels 1360 - 1480).

Zie in hoofdstuk 3 de instrukties: SCREEN, COLOR, PRINT, CHR$, SPRITE$ en PUT SPRITE.

Ter vergelijking is aan het machinetaal-programma het overeenkomstige Basic-programma toegevoegd, regels 1790 en verder.

```
1000 ' ;==================================
1010 ' ;   SPRITE
1020 ' ;
1030 ' ;   1. Machinetaal-testprogramma
1040 ' ;==================================
1050 ' ;
1055 ' ORG 49500
1060 ' INIT32 EQU &H006F
1070 ' INITXT EQU &H006C
1080 ' WRTVRM EQU &H004D
1090 ' FILVRM EQU &H0056
1100 ' CHGCLR EQU &H0062
1110 ' STOP EQU &H00BA
1120 ' ;
1130 ' START CALL INIT32 ;SCREEN 1
1140 ' LD A,15           ;wit
1150 ' LD (&HF3E9),A     ;FORCLR
1160 ' LD A,5            ;blauw
1170 ' LD (&HF3EA),A     ;BAKCLR
1180 ' CALL CHGCLR       ;wijzig kleuren
1190 ' ;
1200 ' LD HL,TXT         ;plaats tekst
1210 ' LD B,6            ;op scherm->
1220 ' LUS2 LD A,(HL)
1230 ' RST &H18          ;print-routine OUTDO
1240 ' INC HL
1250 ' DJNZ LUS2
1260 ' ;
1270 ' LD A,255          ;blok-patroon
1280 ' LD HL,14336       ;begin patroontabel
1290 ' LD BC,32          ;aantal bytes
1300 ' CALL FILVRM       ;vul-routine
1310 ' ;
1320 ' LD A,0            ;teller
1330 ' LD (VAR1),A       ;bewaar A
1340 ' ;
1350 ' LUS0 LD B,255     ;teller
1360 ' LUS1 LD HL,6912
1370 ' ;                  6912=begin attribuut-tabel
1380 ' LD A,100          ;Y-koordinaat
1390 ' CALL WRTVRM       ;VPOKE6912,100
```

```
1400 ' INC HL              ;HL=HL+1
1410 ' LD A,B              ;X-koordinaat
1420 ' CALL WRTVRM         ;VPOKE6913,B
1430 ' INC HL
1440 ' LD A,1              ;patroonnummer
1450 ' CALL WRTVRM         ;VPOKE6914,1
1460 ' INC HL
1470 ' LD A,15             ;kleurnummer
1480 ' CALL WRTVRM         ;VPOKE6915,1
1490 ' CALL STOP           ;Ctrl+Stop?
1500 ' CALL PAUZE          ;vertraag snelheid
1510 ' DJNZ LUS1           ;herhaal tot B=0
1520 ' ;
1530 ' LD A,(VAR1)         ;bekijk teller A
1540 ' INC A
1550 ' CP 3 ;IF A=3 THEN
1560 ' RET Z ;GOTO Basic
1570 ' LD (VAR1),A         ;bewaar A
1580 ' JP LUS0             ;herhaal LUS0
1590 ' ;
1600 ' PAUZE PUSH BC       ;deze routine
1610 ' LD BC,1000          ;telt van 1000
1620 ' LUS3 DEC BC         ;tot 0
1630 ' LD A,B              ;de handeling wordt
1640 ' OR C                ;daardoor vertraagd
1650 ' JP NZ,LUS3
1660 ' POP BC
1670 ' RET
1680 ' ;
1690 ' TXT DM "SPRITE"
1700 ' VAR1 NOP            ;teller
1710 ' EIND END
1720 ' ;
1730 REM
1740 REM   roep MT-routine aan->
1750 REM
1760 (77)  DEFUSR0=49500!:Z=USR0(0)
1762 (212) SCREEN 0:COLOR 1,14
1770 END
1780 REM
1790 REM ================================
```

```
1800 REM   SPRITE
1810 REM
1820 REM   2. Basic-testprogramma
1830 REM ================================
1840 (38)  KEY OFF:DEFINT I-J
1850 (216) SCREEN 1 'INIT32
1860 (95)  COLOR 15,5,1 'wijzig kleuren
1870 (176) PRINT"SPRITE" 'tekst
1880 REM   vul 32 bytes van de
1890 REM   sprite-patroontabel met
1900 REM   de waarde 255 - de sprite
1910 REM   krijgt daardoor een blokvorm
1920 (59)  FOR I=0 TO 31
1930 (146) VPOKE 14336+I,255
1940 (132) NEXT
1950 REM   6912=startadres van de
1960 REM   sprite-attribuut-tabel
1970 (189) FORJ=1 TO 3
1980 (153) FORI=255 TO 1 STEP -1
1990 (160) VPOKE6912,100 'Y-koordinaat
2000 (119) VPOKE6913,I 'X-koordinaat
2010 (64)  VPOKE6914,1 'patroon-nummer
2020 (78)  VPOKE6915,15 'kleurnummer
2030 (205) NEXT I
2040 (206) NEXT J
2050 REM   I varieert, zodat de sprite
2060 REM   over het scherm beweegt
2070 (116) SCREEN 0:KEY ON
2080 (215) COLOR 1,14:END
```

## *6.15 ASCDAC en VARINT: Basic-ROM-routines*

De routines ASCDAC en VARINT. die worden besproken in de hoofdstukken 4.3 en 4.20, maken het mogelijk getallen die in de vorm van ASCII-kode-tekens worden ingevoerd, om te zetten in gehele getallen, die in Z80-registers geplaatst kunnen worden.

Getallen die groter zijn dan 9 kunnen niet zonder meer aan een machinetaal-programma worden doorgegeven. Wanneer we het getal '185' intoetsen, dan doen we in feite niets anders dan het door-

geven van drie tekens aan de computer in ASCII-kode. We kunnen met behulp van een eenvoudige ingreep het ASCII-getal omzetten in het werkelijke getal (door er 48 van af te trekken), maar het is vrij moeilijk om de computer uit te leggen dat het eerst ingevoerde getal in feite de waarde 100 vertegenwoordigt.

Om te voorkomen dat het plezier in het programmeren door dergelijke hindernissen wordt bedorven wordt in dit test-programma getoond hoe we de computer dat werk kunnen laten verrichten. (We moeten daarbij eerlijkheidshalve wel vermelden dat de Basic-ROM-routines niet onder de MSX-standaard vallen, zodat incidenteel afwijkende aanroepadressen kunnen voorkomen.)

In het machinetaal-programma gebruiken we voor de invoer van getallen drie routines:

a. QINLIN - regel 560;
b. ASCDAC - regel 610 en
c. VARINT - regel 660.

Met behulp van die routines kunnen getallen van 1 tot 65535 worden ingetoetst.

De drie routines worden twee keer aangeroepen (regel 310 en regel 380), hetgeen als resultaat twee getallen oplevert. Het eerste getal in registerpaar DE (regel 320); het tweede getal in HL (regel 380).

Met behulp van de 'ADC'-instruktie worden de twee getallen bij elkaar opgeteld (regel 440). Omdat de inhoud van het carry-bit bij het resultaat wordt opgeteld (kenmerk van een 'ADC'- of 'SBC'-bewerking) moeten we er voor zorgen dat de inhoud van dat bit 'nul' is. Dat gebeurt in regel 430 met behulp van de logische bewerking 'OR'.

Er zou ook kunnen worden gekozen voor de kombinatie 'SCF' (set carry flag) en 'CPL' (komplement carry flag). De SCF-instruktie maakt de carry-vlag hoog; de komplement-instruktie 'inverteert' het carry-bit: 'nul' wordt 'een' en 'een' wordt 'nul'. Is het resultaat van de optelling te groot (carry-vlag wordt hoog) dan zal een foutmelding worden gegeven (regel 450).

Het Basic-aanroep-programma (regel 770 - 990) kan worden geRUNd nadat het machinetaal-programma in de computer is geladen. Het print tweemaal een tekst op het scherm (regels 890 en 900) en plaatst ten behoeve van de machinetaal-routine, die in regel 940 wordt aangeroepen, de kursor achter het eerste woord (regel 920).

```
100 ' ;==================================
110 ' ;   ASCDAC+VARINT: TEST-PROGRAMMA
120 ' ;   omzetting ASCII-getallenstring
130 ' ;   in variabelen-notatie plus
140 ' ;   omzetting variabelen in
150 ' ;   gehele getallen
160 ' ;   Machinetaalprogramma
170 ' ;==================================
180 ' ;
190 ' ORG 49500
200 ' DAC2 EQU &HF7F8
210 ' VALTYP EQU &HF663
220 ' STOP EQU &H00BA
230 ' QINLIN EQU &H00B4
240 ' HLPRT EQU &H3412
250 ' LINADR EQU &H4295
260 ' ERROR EQU &H406F
270 ' ASCDAC EQU &H3299
280 ' VARINT EQU &H5439
290 ' CSRX EQU &HF3DD
300 ' ;
310 ' START CALL SUBR ;getal1 in HL
320 ' EX DE,HL        ;SWAP HL,DE
330 ' PUSH DE         ;bewaar DE
340 ' ;
350 ' LD A,8          ;plaats kursor 8 posities
360 ' LD (CSRX),A     ;naar rechts
370 ' ;
380 ' CALL SUBR       ;getal2 in HL
390 ' POP DE          ;herstel DE (getal1 in DE)
400 ' ;
410 ' ;                tel ingevoerde getallen bij
420 ' ;                elkaar op->
430 ' OR A            ;reset Carry-vlag
```

```
440 ' ADC HL,DE        ;HL=HL+DE
450 ' JP C,FOUT
460 ' ;                 IF (HL+DE)>65535 THEN GOTO FOUT
470 ' ;
480 ' LD A,5           ;plaats kursor 5 posities
490 ' LD (CSRX),A      ;naar rechts
500 ' CALL HLPRT       ;print HL
510 ' LD A,2           ;integerkode
520 ' LD (VALTYP),A
530 ' ;
540 ' RET              ;terug naar Basic
550 ' ;
560 ' SUBR CALL QINLIN ;LINEINPUT
570 ' ;                 ingevoerde tekst wordt in
580 ' ;                 buffer BUF geplaatst
590 ' INC HL           ;startadres BUF in HL
600 ' ;
610 ' CALL ASCDAC      ;->
620 ' ;                 string in BUF wordt omgezet
630 ' ;                 in variabelennotatie en in
640 ' ;                 buffer DAC geplaatst
650 ' ;
660 ' CALL VARINT      ;->
670 ' ;                 variabele in DAC wordt omgezet
680 ' ;                 in een geheel getal en in
690 ' ;                 registerpaar HL geplaatst
700 ' RET              ;RETURN
710 ' ;
720 ' FOUT LD E,6      ;OVERFLOW-kode in E
730 ' JP ERROR         ;ga naar foutverwerker
740 ' ;
750 ' EIND END
760 REM
770 REM ===============================
780 REM   Basic-testprogramma
790 REM ===============================
800 REM
810 (100) DEFUSR=49500!
820 (155) CLS:COLOR 1,14
830 (56)  KEY OFF:C=0
840 (4)   STOPON:ON STOP GOSUB 990
```

```
850 REM   twee gehele getallen worden
860 REM   bij elkaar opgeteld
870 (246) IF C>19 THEN GOSUB 960:CLS
880 (28)  C=CSRLIN 'kursor line
890 (165) PRINT"GETAL-1"
900 (166) PRINT"GETAL-2"
910 (99)  PRINT"som="
920 (98)  LOCATE 7,C 'zet kursor terug
930 REM   roep machinetaal-routine aan:
940 (139) Z=USR(0)
950 (154) PRINT:PRINT:GOTO 870
960 REM   pauze->
970 (65)  FOR I=0 TO 500:NEXT:RETURN
980 REM   einde->
990 (248) CLS:KEY ON:END
```

## *6.16 DACHEX: Basic-ROM-routine*

Met behulp van deze routine kan de inhoud van een Z80-registerpaar in hexadecimale vorm op het scherm worden afgedrukt (zie hoofdstuk 4.10).

Het machinetaal-programma, dat in feite een vertaling is van het kleine Basic-programma dat in de regels 630 - 860 is opgenomen, wist het scherm (regel 270) en plaatst de waarde 0 in registerpaar HL (regel 280). Vervolgens wordt binnen een programma-lus de subroutine HEXPRT aangeroepen (regels 300 - 370).

Deze subroutine (regels 430 - 590) bestaat uit de volgende stappen:

a. plaats de inhoud van registerpaar HL in de systeembuffer DAC, en wel op de adressen &HF7F8 en &HF7F9;

b. zet het getal in DAC om in een 'hexadecimale string' met behulp van de ROM-routine DACHEX;

c. print deze string met behulp van de ROM-routine STRPRT;

d. voer een 'carriage-return' en 'linefeed'-opdracht uit met behulp van de ROM-routine CRLF.

Om een foutmelding te voorkomen wordt in de regels 460 en 470 de integerkode 2 in de systeemvariabele VALTYP geplaatst. In de regels 530 - 550 tenslotte wordt de letter 'H' aan de string toegevoegd.

Het programma kan worden beëindigd door het indrukken van de toetsen Ctrl en Stop (regel 300).

```
100 ' ;================================
110 ' ;   DACHEX:  TESTPROGRAMMA
120 ' ;   getallen in hexadecimale vorm
130 ' ;   op het scherm plaatsen
140 ' ;   Machinetaalprogramma
150 ' ;================================
160 ' ;
170 ' ORG 49500
180 ' DAC2 EQU &HF7F8
190 ' VALTYP EQU &HF663
200 ' PRINT EQU &H00A2
210 ' CRLF EQU &H7328
220 ' DACHEX EQU &H3722
230 ' STRPRT EQU &H6678
240 ' CLS EQU &H00C3
250 ' STOP EQU &H00BA
260 ' ;
270 ' START CALL CLS ;wis scherm
280 ' LD HL,0        ;beginwaarde=0
290 ' ;
300 ' LUS CALL STOP  ;Ctrl+Stop?
310 ' ;               voer subroutine uit->
320 ' CALL HEXPRT
330 ' ;               verhoog HL->
340 ' INC HL         ;HL=HL+1
350 ' ;               herhaal tot Ctrl+Stop-toetsen
360 ' ;               worden ingedrukt->
370 ' JP LUS
380 ' ;
390 ' ;               HEXPRT is de subroutine die
400 ' ;               inhoud HL in hexadecimale vorm
410 ' ;               op het scherm zet->
420 ' ;
```

```
430 ' HEXPRT PUSH HL ;bewaar HL
440 ' LD (DAC2),HL   ;plaats inhoud
450 ' ;               HL in de systeembuffer DAC
460 ' LD A,2         ;en zet integergetalkode
470 ' LD (VALTYP),A  ;in VALTYP
480 ' ;
490 ' CALL DACHEX    ;zet inhoud HL om
500 ' ;               in een hexadecimale string
510 ' CALL STRPRT    ;en print de string
520 ' ;
530 ' LD A,72        ;plaats ASCII-waarde "H"
540 ' ;               in register A en
550 ' CALL PRINT     ;print "H"
560 ' CALL CRLF      ;Carr.Return+Linefeed
570 ' ;
580 ' POP HL         ;herstel HL
590 ' RET            ;terug naar hoofdprogramma
600 ' ;
610 ' EIND END
620 ' ;
630 REM routine in werking stellen->
640 DEFUSR0=49500!:Z=USR0(0)
650 END ' einde test-routine!
660 REM
670 REM
680 REM ================================
690 REM   HETZELFDE PROGRAMMA IN BASIC:
700 REM ================================
710 REM
720 (160) CLS 'wis scherm
730 (151) HL=0
740 REM   LUS->
750 (121) GOSUB 820
760 (31)  HL=HL+1
770 (67)  GOTO 750
780 REM
790 REM ================================
800 REM   subroutine->
810 REM ================================
820 (81)  PRINTHEX$(HL); 'zet HL om
830 (91)  PRINT"H"     ; 'voeg "H" toe
```

```
840 (146) PRINT 'Carr.Return/Linefeed
850 (143) RETURN
860 (130) END
```

## *6.17 DIVDAC: Basic-ROM-routine*

Het delen van getallen is een ingewikkelde zaak. Omdat de Z80-processor geen deel-instrukties kent (behalve dan de schuif- en roteerinstrukties) wordt in dit boek een Basic-routine besproken (zie hoofdstuk 4.12) die twee gehele getallen deelt en het resultaat in de systeembuffer DAC plaatst.

In het test-programma dat we hier behandelen wordt gebruik gemaakt van deze deel-routine. De getallen worden met behulp van een Basic-aanroep-programma doorgegeven aan de machinetaal-routine (regels 600 en 630), waarna de deel-routine in werking wordt gesteld (regel 680).

De deel-routine bestaat uit de volgende stappen:

a. roep de deel-routine DIVDAC aan;

b. kontroleer of de ingevoerde getallen positief zijn (d.w.z. niet groter dan 32767) - hiervoor wordt de ROM-routine DACSGN gebruikt;

c. zet het resultaat dat in de systeembuffer is geplaatst om in een geheel getal met behulp van routine VARINT;

d. print dit getal met behulp van de routine HLPRT.

```
100 ' ;================================
110 ' ;   DIVDAC: TESTPROGRAMMA
120 ' ;   deel twee getallen
130 ' ;   Machinetaalprogramma
140 ' ;================================
145 ' ;
150 ' ORG 49500
160 ' VALTYP EQU &HF663
170 ' DIVDAC EQU &H4DB8
```

```
180 ' VARINT EQU &H5439
190 ' HLPRT EQU &H3412
200 ' ERROR EQU &H406F
210 ' DAC EQU &HF7F6
220 ' ;
230 ' START LD DE,(VAR1)
240 ' LD HL,(VAR2)
250 ' CALL DIVDAC  ;DE/HL
260 ' ;             inhoud DE wordt gedeeld door
270 ' ;             inhoud HL
280 ' ;             resultaat wordt in DAC gezet
290 ' ;
300 ' CALL &H2EA1  ;DACSGN-tekencheck
310 ' LD E,6       ;zet OVERFLOW-kode in E
320 ' JP M,ERROR   ;ga naar foutverwerker
330 ' ;             als een van de getallen negatief
340 ' ;             is (groter dan 32767)
350 ' ;
360 ' CALL VARINT
370 ' ;             variabele in DAC wordt omgezet
380 ' ;             in geheel getal en in HL
390 ' ;             geplaatst
400 ' CALL HLPRT
410 ' ;             zet getal in HL om in string
420 ' ;             en print string
430 ' LD A,2       ;integer-kode
440 ' LD (VALTYP),A
450 ' RET
460 ' ;
470 ' VAR1 DS 2
480 ' VAR2 DS 2
490 ' EIND END
500 ' ;
510 REM ================================
520 REM   Basic-testprogramma
530 REM ================================
540 REM
550 (52)  CLS:COLOR 1,7,7
560 (26)  KEY OFF:STOP ON
570 (158) ON STOP GOSUB 730
580 (47)  INPUT"GETAL 1";V1
```

```
590 (225) B=INT(V1/256):A=V1-256*B
600 (93)  POKE 49530!,A:POKE 49531!,B
610 (49)  INPUT"GETAL 2";V2
620 (227) B=INT(V2/256):A=V2-256*B
630 (156) POKE 49532!,A:POKE 49533!,B
640 (146) PRINT
650 (80)  PRINT "De deel-routine ";
660 (214) PRINT "plaatst als"
670 (207) PRINT "resultaat de waarde ";
680 (77)  DEFUSR0=49500!:Z=USR0(0)
690 (146) PRINT
700 (21)  PRINT"in registerpaar HL"
710 (146) PRINT
720 (199) GOTO 580
730 (248) CLS:KEY ON:END
```

### *6.18 LINADR: Basic-ROM-routine*

Het opzoeken van een programma-regel binnen een Basic-programma is een eenvoudige zaak wanneer men gebruik maakt van de routine LINADR (zie hoofdstuk 4.15).

Het kleine test-programma dat hier staat afgedrukt laat zien hoe een en ander in zijn werk gaat. Het regelnummer dat opgezocht moet worden wordt via een Basic-aanroep-programma als parameter (variabele R) doorgegeven aan de machinetaal-routine (regels 480 - 520). De routine LINADR haalt het regelnummer uit systeembuffer DAC en gaat op zoek.

Wordt het regelnummer gevonden dan zal de zero-vlag hoog (1) worden gemaakt; in het andere geval is de zero-vlag laag (0) en kan een fout-routine in werking worden gesteld (regel 260). Omdat de routine het startadres van de regel in registerpaar BC plaatst (wanneer een regel is gevonden), wordt de inhoud van BC in registerpaar HL geladen in de regels 280 en 290. Dat adres kan nu met behulp van de print-routine HLPRT op het scherm worden afgedrukt.

```
100 ' ;=================================
110 ' ;   LINADR: TESTPROGRAMMA
115 ' ;   zoek een programmaregel
120 ' ;   Machinetaalprogramma
130 ' ;=================================
140 ' ;
150 ' ORG 49500
160 ' LINADR EQU &H4295
170 ' ERROR EQU &H406F
180 ' HLPRT EQU &H3412
190 ' VALTYP EQU &HF663
200 ' DAC2 EQU &HF7F8
210 ' ;
220 ' START LD DE,(DAC2)
230 ' ;                de geheugenplaatsen F7F8 en
240 ' ;                F7F9 bevatten het regelnummer
250 ' CALL LINADR     ;regelzoekroutine
260 ' JP NZ,FOUT      ;wordt regel niet
270 ' ;                gevonden dan foutmelding
280 ' LD H,B          ;plaats inhoud BC
290 ' LD L,C          ;in registerpaar HL
300 ' CALL HLPRT      ;print HL
310 ' LD A,2          ;integer-kode
320 ' LD (VALTYP),A
330 ' RET             ;naar Basic
340 ' ;
350 ' FOUT LD E,8     ;foutkode 8 in E
360 '                 ;8=undefined line number
370 ' JP ERROR        ;naar foutverwerker
380 ' ;
390 ' EIND END
400 ' ;
410 REM ================================
420 REM  Basic-testprogramma
430 REM ================================
440 (39)  CLS:KEY OFF:COLOR 1,7,7
450 (100) DEFUSR=49500!
460 (44)  STOP ON:ON STOP GOSUB 540
470 (59)  DEFINT A-Z 'gehele getallen
480 (5)   INPUT"WELK REGEL-NUMMER";R
490 (144) PRINT"startadres:";
```

```
500 REM   geef het regelnummer door aan
510 REM   de machinetaalroutine->
520 (204) Z=USR(R)
530 (254) PRINT:PRINT:GOTO 480
540 (248) CLS:KEY ON:END
```

### *6.19 MLPDAC: Basic-ROM-routine*

In hoofdstuk 4.16 wordt uiteengezet hoe twee getallen met elkaar kunnen worden vermenigvuldigd met behulp van de routine MLPDAC. Het test-programma laat zien hoe twee getallen vanuit een Basic-aanroep-programma kunnen worden doorgegeven aan de machinetaal-routine, die na de vermenigvuldiging te hebben uitgevoerd het resultaat op het scherm afdrukt.

De volgende stappen worden gezet:

a. vermenigvuldig de twee getallen die uit het geheugen zijn gehaald met behulp van routine MLPDAC;

b. kontroleer of positieve getallen werden ingevoerd met behulp van routine DACSGN;

c. zet de variabele in de systeembuffer DAC om in een ASCII-getallen-string;

d. print de string met behulp van routine STRPRT.

```
100 ' ;================================
110 ' ;   MPLDAC: TESTPROGRAMMA
120 ' ;   vermenigvuldig twee getallen
130 ' ;   Machinetaalprogramma
140 ' ;================================
150 ' ;
160 ' ORG 49500
170 ' VALTYP EQU &HF663
180 ' MLPDAC EQU &H3193
190 ' DACASC EQU &H3425
200 ' STRPRT EQU &H6678
210 ' DACSGN EQU &H2EA1
```

```
220 ' ERROR EQU &H406F
230 ' BEEP EQU &H00C0
240 ' ;
250 ' START LD DE,(VAR1)
260 ' LD HL,(VAR2)
270 ' CALL MLPDAC     ;->
280 ' ;                inhoud DE wordt vermenigvuldigd
290 ' ;                met inhoud HL
300 ' ;                resultaat wordt in DAC gezet
310 ' ;                DE of HL mogen niet negatief
320 ' ;                (d.w.z groter dan 32767) zijn
330 ' ;
340 ' CALL DACSGN     ;kontroleer teken
350 ' LD E,6          ;Overflow-kode in E
360 ' JP M,ERROR      ;negatief->foutmelding
370 ' ;
380 ' CALL DACASC
390 ' ;                variabele in DAC wordt omgezet
400 ' ;                in ASCII-string
410 ' ;
420 ' CALL STRPRT
430 ' ;                ASCII-string wordt afedrukt
440 ' ;                op het scherm
450 ' LD A,2          ;integer-kode
460 ' LD (VALTYP),A
470 ' RET
480 ' ;
490 ' VAR1 DS 2
500 ' VAR2 DS 2
510 ' EIND END
520 ' ;
530 REM ================================
540 REM   Basic-testprogramma
550 REM ================================
560 REM
570 (52)  CLS:COLOR 1,7,7
580 (26)  KEY OFF:STOP ON
590 (168) ON STOP GOSUB 740
600 (47)  INPUT"GETAL 1";V1
610 (225) B=INT(V1/256):A=V1-256*B
620 (93)  POKE 49530!,A:POKE 49531!,B
```

```
630 (49)  INPUT"GETAL 2";V2
640 (227) B=INT(V2/256):A=V2-256*B
650 (156) POKE 49532!,A:POKE 49533!,B
660 (146) PRINT
670 (214) PRINT "de test-routine";
680 (21)  PRINT "plaatst als"
690 (47)  PRINT "resultaat de waarde";
700 (77)  DEFUSR0=49500!:Z=USR0(0)
710 (125) PRINT" op het scherm"
720 (146) PRINT
730 (243) GOTO 600
740 (248) CLS:KEY ON:END
```

## *6.20 VARADR: Basic-ROM-routine*

Het opzoeken van een variabele kan in een Basic-programma gebeuren met behulp van de instruktie VARPRT. De routine VARADR kan binnen een machinetaal-programma hetzelfde doen. Voorwaarde is dat de naam van de variabele (in hoofdletters) met behulp van de pseudo-instruktie 'DM' (define message) in het bronprogramma wordt opgenomen. In het test-programma wordt de integer-variabele A% opgezocht (zie regel 470).

Het adres van de geheugenlokaties die de naam bevatten wordt doorgegeven aan de zoekroutine (regel 210), die op zijn beurt het adres van de eerste byte van het geheugengebied dat de inhoud van de variabele bevat in registerpaar DE plaatst.

Omdat het geheugengebiedje dat de inhoud van de variabele bevat wordt voorafgegaan door drie bytes die het type van de variabele en de naam van de variabele aangeven, wordt in de regels 280 en 290 de inhoud van registerpaar DE doorgegeven aan het indexregister IX.

Nadat het adres met behulp van routine HLPRT is geprint (regels 300 en 310) en een scheidingsteken is toegevoegd (regels 320 en 330) wordt het type van de variabele gekontroleerd. Daarbij wordt gebruik gemaakt van de speciale mogelijkheden van de indexregisters: namelijk het gebruiken van verplaatsingswaarden (zie regel 350).

Is de gevonden kode niet gelijk aan 2 (integer-getal-kode) dan wordt een return-instruktie uitgevoerd (regel 370). In het andere geval wordt de inhoud van de variabele in registerpaar HL geladen en afgedrukt op het scherm (regels 400 - 420).

Om het zoeken van andere variabelen mogelijk te maken wordt in het Basic-aanroep-programma getoond hoe met behulp van een 'POKE'-kommando de naam van de variabele (in de vorm van een ASCII-kode-getal) kan worden doorgegeven aan de machinetaalroutine (regels 800 - 910).

```
100 ' ;==================================
110 ' ;   VARADR: TESTPROGRAMMA
120 ' ;   zoek een variabele
130 ' ;   Machinetaalroutine
140 ' ;==================================
150 ' ;
160 ' ORG 49500
170 ' VARADR EQU &H5EA4
180 ' HLPRT EQU &H3412
190 ' VALTYP EQU &HF663
200 ' ;
210 ' LD HL,VARNM     ;plaats het adres
220 ' ;                van de naam van de variabele
230 ' ;                in registerpaar HL
240 ' CALL VARADR     ;zoekroutine
250 ' ;                plaatst adres inhoud variabele
260 ' ;                in registerpaar DE
270 ' ;
280 ' PUSH DE         ;maak IX gelijk
290 ' POP IX          ;aan DE
300 ' EX DE,HL        ;maak HL gelijk aan DE
310 ' CALL HLPRT      ;print het adres
320 ' LD A,47         ;/ teken
330 ' RST &H18        ;tussenvoegen
340 ' ;
350 ' LD A,(IX-3)     ;bepaal var-type
360 ' CP 2            ;2=integer-kode
370 ' RET NZ          ;IF A<>2 THEN RETURN
380 ' ;
390 ' ;                plaats inhoud variabele in HL->
```

```
400 ' LD L,(IX+0)     ;LSB-lage byte
410 ' LD H,(IX+1)     ;MSB-hoge byte
420 ' CALL HLPRT      ;print HL
430 ' LD A,2          ;integer-kode
440 ' LD (VALTYP),A
450 ' RET             ;naar Basic
460 ' ;
470 ' VARNM DM "A%"   ;variabele
480 ' NOP
490 ' EIND END
500 ' ;
510 REM ================================
520 REM   Basic-testprogramma
530 REM ================================
540 (62)  CLS:DEFUSR=49500!
550 (9)   KEY OFF:COLOR 1,14,14
560 (9)   STOP ON:ON STOP GOSUB 740
570 (51)  V$(1)="A%":V$(2)="B%"
580 (127) V$(3)="C%"
590 (136) PRINT:FOR I=1 TO 3
600 (180) PRINT"WELKE WAARDE HEEFT ";
610 (245) PRINT"VARIABELE ";V$(I);
620 (213) ON I GOSUB 800,840,880
630 (42)  PRINT:PRINT"ADRES en INHOUD:";
640 REM   roep machinetaalroutine aan
650 (157) Z=USR0(0)
660 (101) PRINT:PRINT:NEXT I
670 (77)  PRINT"dezelde adressen ";
680 (252) PRINT"kunnen worden"
690 (224) PRINT"gevonden met de Basic";
700 (10)  PRINT"-instrukties:"
710 (133) PRINT"PRINT VARPTR";
720 (16)  PRINT"(variabele)+65536"
730 (146) PRINT
740 (30)  KEY ON:END
750 REM ================================
760 REM   subroutines->
770 REM   wijzigen naam variabele
780 REM   in machinetaalprogramma
790 REM ================================
800 REM   var1=A%
```

```
810 (236) INPUT A%
820 (136) POKE 49537!,65 'VARNM
830 (143) RETURN
840 REM   var2=B%
850 (237) INPUT B%
860 (137) POKE 49537!,66
870 (143) RETURN
880 REM   var3=C%
890 (238) INPUT C%
900 (138) POKE 49537!,67
910 (143) RETURN
```

Voor veel MSX gebruikers is de overstap van Basic naar machinetaal te groot. Zij worden plotseling gekonfronteerd met een volledig nieuw jargon, dat niets met het vertrouwde MSX Basic te maken lijkt te hebben. Daar komt bij, dat men genoodzaakt is een vertaalprogramma aan te schaffen dat de machinetaal-instrukties omzet in voor de computer begrijpelijke taal: de assembler.

Dit boek, dat een brug wil slaan tussen MSX Basic en machinetaal, maakt daarom gebruik van de nieuwe nederlandse assembler FLASH (verschenen bij dezelfde uitgever).
Hoewel het bezit van deze assembler geen absolute noodzaak is om de stof te begrijpen, is de aanschaf ervan voor beginnende gebruikers sterk aan te raden.

Om de MSX gebruiker optimaal van dienst te zijn, wordt voortdurend verwezen naar het vertrouwde MSX Basic; Z80 assemblerinstrukties worden met behulp van Basic testprogramma's verhelderd en letterlijk konkreet gemaakt, door de resultaten op het scherm te tonen.

Het was de opzet van de auteur, een praktijk-boek te schrijven, dat de gebruiker in staat stelt zélf kleine routines te schrijven voor gebruik in Basic programma's. Ingewikkelde routines, voorbehouden aan professionele programma's, komen in het boek dan ook niet aan de orde.

**ISBN 90 6398 669 6** **uitgeverij Stark-Texel b.v.**